UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA E MECÂNICA
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

GUILHERME ZASYEKI MACHADO

DESENVOLVIMENTO DE UM KIT DIDÁTICO PARA ESTUDO DE ROBÓTICA: VEÍCULO REMOTAMENTE CONTROLADO ATRAVÉS DA PLATAFORMA ROS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2013

GUILHERME ZASYEKI MACHADO

DESENVOLVIMENTO DE UM KIT DIDÁTICO PARA ESTUDO DE ROBÓTICA: VEÍCULO REMOTAMENTE CONTROLADO ATRAVÉS DA PLATAFORMA ROS

Trabalho de Conclusão de Curso, apresentado ao Curso Superior de Tecnologia em Mecatrônica Industrial, dos Departamentos Acadêmicos de Eletrônica e Mecânica, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Tecnólogo.

Orientador: Prof. M. Sc. Juliano Mourão Vieira

CURITIBA
2013

TERMO DE APROVAÇÃO

GUILHERME ZASYEKI MACHADO

DESENVOLVIMENTO DE UM KIT DIDÁTICO PARA ESTUDO DE ROBÓTICA: VEÍCULO REMOTAMENTE CONTROLADO ATRAVÉS DA PLATAFORMA ROS

Este trabalho de conclusão de curso foi apresentado no dia 10 de outubro de 2013, como requisito parcial para obtenção do título de Tecnólogo em Mecatrônica Industrial, outorgado pela Universidade Tecnológica Federal do Paraná. O aluno foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

____
Prof. Dr. Milton Luiz Polli
Coordenador de Curso
Departamento Acadêmico de Mecânica

____
Prof. Esp. Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

____
Prof. Dr. André Schneider de Oliveira
UTFPR

____
Prof. Dr. Luís Alberto Lucas
UTFPR

____
Prof. M.Sc. Juliano Mourão Vieira
Orientador - UTFPR

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”

AGRADECIMENTOS

Inicialmente gostaria de agradecer ao meu orientador Prof. Me. Juliano Mourão Vieira, pela paciência e compreensão no desenvolvimento do trabalho.

Aos meus familiares pela paciência.

A meu irmão, pela cooperação.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

RESUMO

MACHADO, Guilherme Zasyeki. **Desenvolvimento de um kit didático para estudo de robótica**: veículo remotamente controlado através da plataforma ROS. 2013. 76 f. Trabalho de Conclusão de Curso (Tecnologia em Mecatrônica Industrial) - Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

O trabalho consiste em desenvolver um veículo remotamente controlado através do *framework* de robótica ROS e, através de tutoriais, visa esclarecer conceitos básicos da plataforma. Para isso utilizou-se placas de desenvolvimento baseadas na plataforma *Arduino*, um veículo *Wild Thumper* de 6 rodas, uma *Panda Board* e um smartphone *Android*.

Palavras-chave: ROS. Robótica. Wild Thumper. Tele operação. Arduino.

// ABSTRACT

MACHADO, Guilherme Zasyeki. **Development of a teaching kit to study robotics:** remotely controlled vehicle through the ROS platform. 2013. 76 f. Trabalho de Conclusão de Curso (Tecnologia em Mecatrônica Industrial) - Federal Technology University - Parana. Curitiba, 2013.

The focus of this work is to develop a remotely controlled vehicle using the robotics framework ROS and clarify basic concepts of the platform through tutorials. Development boards based on the Arduino platform were used, together with a 6 wheel Wild Thumper vehicle, a Panda Board and an Android smartphone.

Keywords: ROS. Robotic. Wild Thumper. Tele operation. Arduino.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

ABIPTI	Associação Brasileira das Instituições de Pesquisa Tecnológica
ARM	Advanced RISC Machine
POSIX	Portable Operating System Interface
RISC	Reduced Instruction Set Computer
ROS	Robot Operating System
SO	Sistema Operacional
URI	Uniform Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos diversos *frameworks* voltados para robótica (um conjunto de código voltados para o desenvolvimento) vêm sendo desenvolvidos, como o Player, YARP, Orocos, CARMEM, Orca, MOOS e Microsoft Robotics Studio (ROS INTRODUCTION, 2012). O objetivo dessas ferramentas é facilitar a integração de diversos componentes, através de uma interface de comunicação bem definida entre eles.

Nesta monografia será abordado o *framework* ROS (*Robot Operating System*), que além de facilitar o desenvolvimento de aplicações para robótica, com uma interface que permite a troca de mensagens entre nós de uma rede distribuída, também tem por objetivo permitir o reuso de códigos existentes, viabilizando a criação de aplicações com maior facilidade e rapidez.

No decorrer do trabalho, desenvolveu-se diversos manuais de instalação e utilização que viabilizam a reprodução e aprendizado do ROS e a interface entre diversos sistemas. Com isso em mente, foi desenvolvida uma aplicação de controle remoto de um veículo através de uma rede Wi-Fi com o auxílio do *framework*, com sinais de controle gerados à partir de um *smartphone Android*.

1.1 PROBLEMA

Inicialmente quando uma empresa ou universidade financiava o desenvolvimento de um robô, o processo era complexo e lento, pois boa parte dos códigos necessários para manipular o robô era novo e pouco código era reaproveitado, devido às aplicações específicas em que eram empregados. Com o tempo, desenvolveram-se plataformas (*frameworks*) visando acelerar este processo.

Esses *frameworks* são projetados, em sua maioria, em centros de ensino ou empresas de ponta, que possuem viabilidade técnica e financeira para tal. Apesar de facilitarem o desenvolvimento e integração de novos componentes aos robôs, estes *frameworks* ainda possuem uma curva de aprendizado muito longa, pois seu uso requer conhecimentos em diversas áreas, não sendo de trivial utilização.

No Brasil, comparativamente a centros de pesquisa de outros países, ainda são pouco utilizados, muitas vezes por total desconhecimento das opções existentes.

1.2 JUSTIFICATIVA

O ROS é uma plataforma com grade aceitação no meio acadêmico e científico, principalmente no exterior, visto o número de módulos existentes (2000+) (ROS LIST, 2012). É utilizado em diversos cursos de robótica em universidades reconhecidas como *Stanford University*, *Tokyo University e Cornell University* (ROS COURSES, 2012). No Brasil o framework não é tão utilizado como em outros lugares do mundo, apesar de, teoricamente, facilitar o desenvolvimento das aplicações para a robótica.

O *framework* vem sendo tema de muitos artigos recentemente, com 6 publicações, em 2010, no ICRA (*International Conference on Robotics and Automation*), uma das conferências mais importantes no meio, e 5 em publicações em 2011 (ROS PAPERS, 2012).

A maior vantagem ao se empregar o ROS é a facilidade com que os blocos podem ser integrados, pois ele possui uma interface bem definida para a comunicação entre os componentes. Como exemplo, seja um robô que já possui o módulo de controle integrado ao ROS ao qual pretende-se adicionar uma câmera estéreo. Se esta já possui o driver, a integração é feita através de uma simples linha de comando. Outra vantagem do ROS é que ele é *Open Source*, ou seja, todos têm acesso ao seu código fonte e podem contribuir com ele. Porém o aprendizado da plataforma não é trivial e por isto este trabalho se propõe a desenvolver uma aplicação que viabilize o aprendizado de forma um pouco mais didática.

1.3 OBJETIVOS

O trabalho consiste em agrupar diversos componentes eletrônicos e de software e fazê-los trabalhar em conjunto para o controle de um veículo remotamente, com isso desenvolvendo materiais que facilitem o aprendizado da principal ferramenta utilizada no projeto, o ROS.

1.3.1 Objetivo geral

Integrar diversos componentes eletrônicos e de software (ROS e softwares embarcados) e utilizá-los de forma coerente, visando o correto uso da plataforma de desenvolvimento (ROS), demonstrando suas facilidades e utilidade de forma didática, desenvolvendo tutoriais e manuais para tal.

1.3.2 Objetivos específicos

- Desenvolver a interface necessária entre ROS e a placa de controle do veículo remoto utilizando a arquitetura de nós distribuídos do ROS através da rede Wi-Fi;
- Agregar conhecimento extensivo e concreto no *framework* ROS utilizando apenas softwares *Open Source*;
- Desenvolver tutoriais e manuais didáticos que facilitem o aprendizado da plataforma ROS;
- Controlar o veículo através da aplicação de tele operação do ROS para *Android* (WILLOW TELEOP, 2013);

1.4 VISÃO GERAL

O projeto é composto de três placas, sendo uma para controle dos motores, baseada no modelo Arduino Nano ATmega168, uma placa com maior capacidade de processamento para realizar a interface entre o controlador dos motores e o ROS e a *Panda Board* propriamente dita que executa o *core* do ROS. Todas são fisicamente colocadas no veículo (*Wild Thumper*) para que se possa controlá-lo remotamente. O controle remoto do veículo é realizado com o auxílio de *smartphone* com o sistema operacional (SO) *Android*, que se comunica com o ROS core com o auxílio de uma aplicação através de uma conexão Wi-Fi. A placa de controle dos motores é gerenciada através de uma conexão I2C, com os comandos de velocidade, direção e sentido processados através do *Arduino Uno* que realiza a interface com ROS. Também faz-se necessário a presença de um roteador Wi-Fi, para criação do grafo de comunicação entre os nós ROS.

O diagrama em blocos com a estrutura lógica do trabalho pode ser observado na figura 1. Esta figura demonstra os três nós em execução do ROS, sendo eles o core, para o gerenciamento de todo o sistema, a aplicação *Android*, com a geração dos comandos de direção e sentido, e o cliente, o qual realiza o papel de ponte para que o *Arduino Uno* possa se conectar a rede criada pelo ROS. Nela também é possível observar a ligação entre o *Arduino Uno* e a placa de gerenciamento dos motores: o *Arduino Uno*, por ter um hardware mais robusto em relação ao controlador dos motores, realiza a interface com o ROS através de uma conexão serial, processa os dados e envia os sinais de controle via conexão I2C para a placa de controle dos motores. O ROS core e o ROS *python client* são executados sobre o sistema operacional Ubuntu, baseado no Linux, o qual provê as funcionalidades básicas para o *framework*.
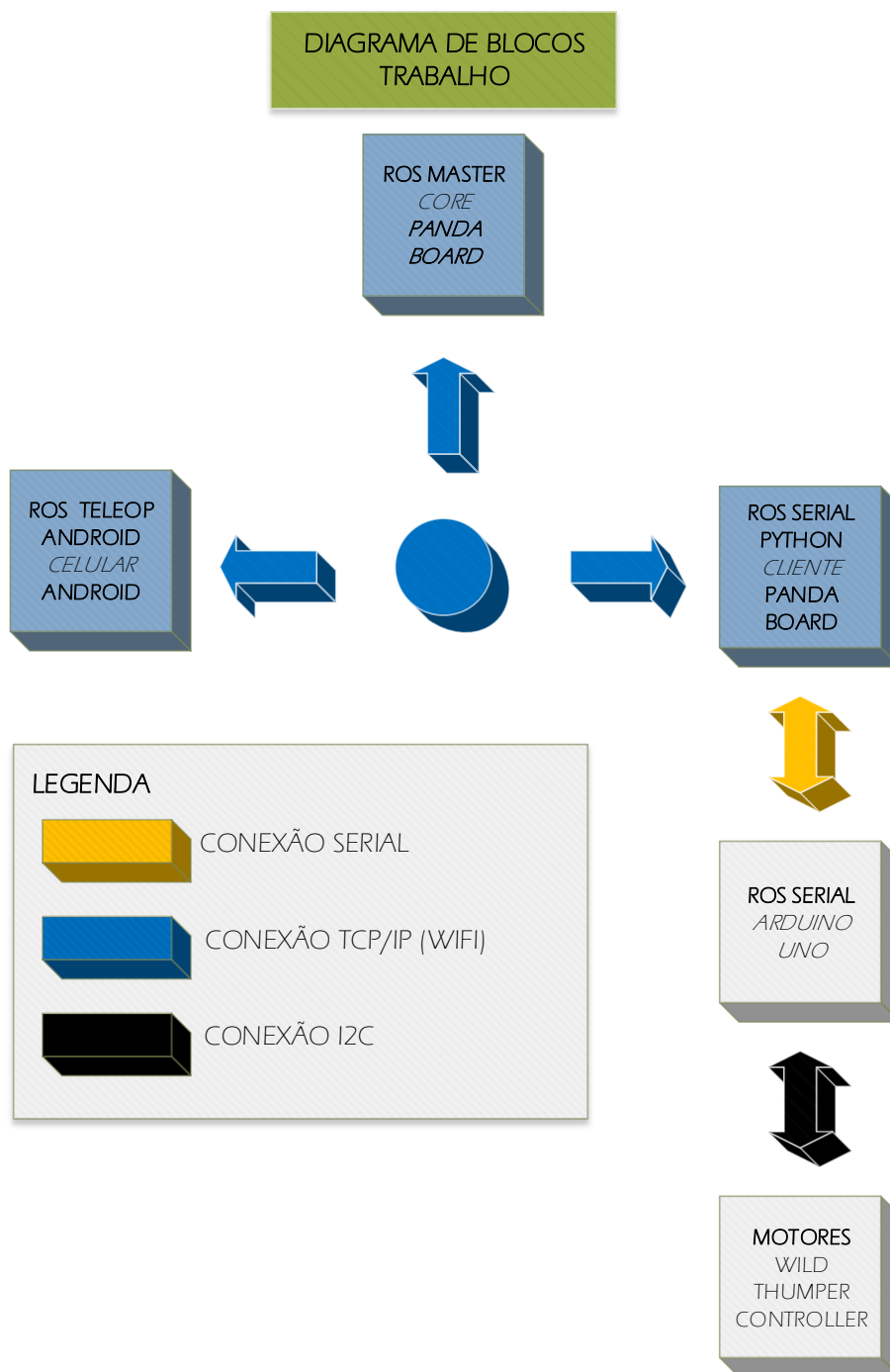
Figura 1 - Diagrama de blocos representando as interações entre os componentes do projeto

Fonte: Autoria Própria.

Para que fosse possível reproduzir com certa facilidade os procedimentos desenvolvidos no decorrer da atividade, criou-se uma série de tutoriais e manuais de utilização, dentre eles:

- Instalação do GNU/Linux Ubuntu 12.04 na *Panda Board*;

- Configuração da *Panda Board* para que todos os recursos necessários funcionem (ex.: Wi-Fi);
- Instalação/Compilação e configuração do ROS na *Panda Board*;
- Montagem e ligações entre as placas e o *Wild Thumper*;
- Manual de utilização básico do ROS, contendo fundamentação da plataforma e comandos básicos;
- Descrição dos procedimentos necessários para reprodução do trabalho;

A relação dos materiais utilizados no trabalho pode ser observada na tabela 1. Esta contém apenas os recursos físicos utilizados. O orçamento total do trabalho pode ser encontrado no apêndice A.

LISTA MATERIAL	
Envolvimento direto com o trabalho	
NOME	DESCRIÇÃO
Wild Thumper	**Veículo voltado para robótica**
Placa controladora Wild Thumper	**Interface de controle dos motores do veículo**
Placa Arduino Uno	**Interface entre controlador WildThumper e Pandaboard**
Panda Board	**Placa central (*node master* ROS) realiza a interface com dispositivo móvel e Arduino Uno.**
Bateria 7.2V 5A	**Alimentação dos motores do WildThumper**
Celular Android	**Enviar sinais de controle para o controlador via ROS**
Cabos de alimentação	**Utilizados para alimentação dos circuitos e motores**
Cabo USB A - USB B	**Ligação entre a Panda Board e o Arduino Uno**
Par de fios com ponta molex	**Ligação I2C entre controlador Wild Thumper e Arduino Uno**
Roteador Wi-Fi	**Para criação da rede TCP/IP distribuída do ROS**
Envolvimento indireto com o trabalho	
NOME	DESCRIÇÃO
Gravador cartão SD	**Gravar a imagem que será utilizada na Panda Board**
Conversor USB-Serial	**Verificação do andamento de instalação**
Cabo USB-miniUSB	**Alimentação da Panda Board**
Cartão SD	**Armazenamento de SO da Panda Board (Obs: recomendável 16 GB)**

Quadro 1 - Lista de materiais que será utilizado no trabalho.

Fonte: Autoria Própria.

1.5 ESTRUTURA DO TRABALHO

Nos capítulos seguintes será descrito em detalhes o desenvolvimento do trabalho. No capítulo 2 serão abordados trabalhos relacionados a este, alguns relacionados pela utilização da plataforma de robótica ROS, outros pela utilização do Wild *Thumper*. No capítulo 3, será abordada a metodologia utilizada no desenvolvimento deste trabalho, com uma descrição detalhada dos procedimentos utilizados. No capítulo 4 serão apresentados alguns testes e resultados obtidos com a utilização do ROS e plataformas embarcadas. No capítulo 5 será apresentada a conclusão do trabalho, assim como sugestões de trabalhos futuros.

2 EMBASAMENTO TEÓRICO

Apesar do ROS ser uma plataforma recente, com sua primeira versão oficial sendo disponibilizada em meados de 2010, é possível observar diversas aplicações práticas da plataforma. A maior parte destes projetos é baseada em dois robôs disponibilizados pela principal mantenedora do projeto ROS, a *Willow Garage*; são eles:

- ***Turtlebot***: um robô de baixo custo que conta com uma câmera, capaz de gerar imagens com perspectiva de profundidade do ambiente e uma base móvel, sendo muito utilizado em projetos cujo objetivo é mapear ambientes e traçar rotas (TURTLE ROS, 2013).
- ***PR2***: este robô possui diversos recursos, todos desenvolvidos a partir do ROS, como: navegação, mapeamento, tele operação e simuladores. Diversos trabalhos voltados a percepção robótica vêm sendo desenvolvidos com esta plataforma, como é o caso de (CHITTA DOORS, 2010) e (CHITTA TACTILE, 2010), os quais trabalham o tato de objetos com a utilização do robô.

Uma das atuais vertentes de pesquisa do ROS é a aplicação industrial da plataforma. Para isso criou-se um consórcio onde diversas empresas podem financiar e se beneficiar da pesquisa e desenvolvimento gerado pelo grupo. Demonstrações práticas e informações podem ser visualizadas em (ROS INDUSTRIAL, 2013).

Um dos grupos de pesquisa que utiliza o ROS com o *Wild Thumper*, é o Team Hector Darmstadt (TEAM HECTOR, 2013), o qual tem por objetivo desenvolver técnicas e fundamentos para a geração de uma rede de colaboração, onde são aplicadas uma grande variedade de sensores e atuadores visando o monitoramento, busca e resgate autônomo. O grupo já desenvolveu diversos módulos para a plataforma do ROS, como pode ser observado em (ROS HECTOR, 2013). Um dos robôs desenvolvidos pelo grupo é o *Hector Lightweight UGV*, baseado no *Wild Thumper* e que participou da competição *RoboCup Rescue Arena* (ROBOCUP, 2013), utilizando o módulo de tele operação do ROS.

2.1 WILD THUMPER

É uma plataforma de entretenimento desenvolvida pela *Dagu Manufactures* com intuito educacional para aprendizado de robótica (WILD, 2012). O kit é composto por um veículo para todos os terrenos e uma placa de controle, vendidos separadamente.
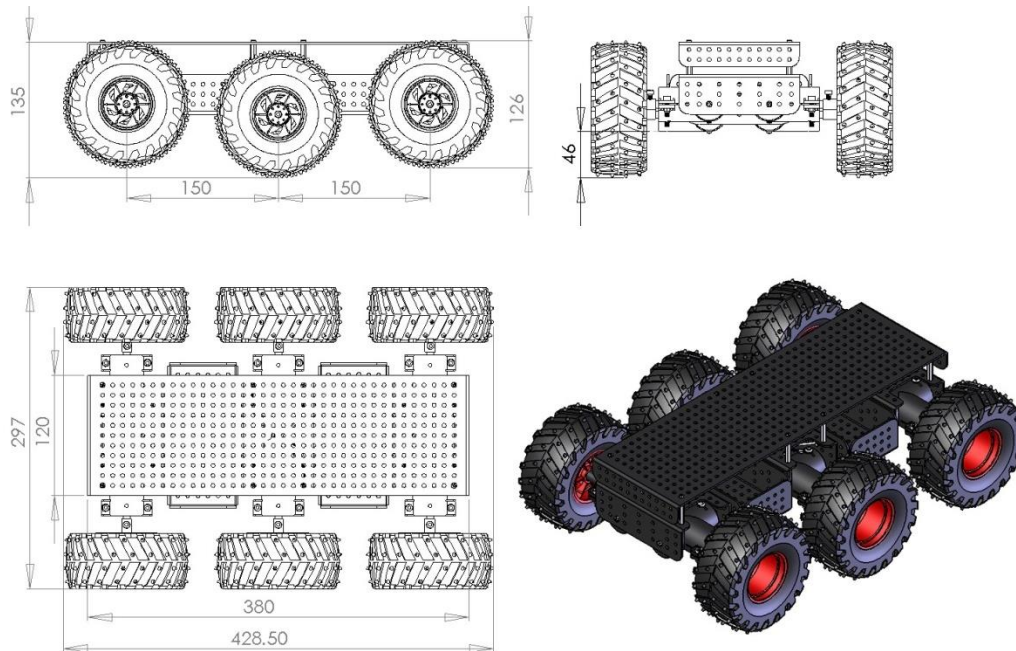
Figura 2 - Visão geral do veículo utilizado no projeto
Fonte: WILD (2012).

O veículo utilizado possui 6 rodas (figura 2) e foi desenvolvido para ser utilizado em todos os tipos de terreno, com uma estrutura bem versátil que possibilita a instalação de equipamentos em sua superfície, possuindo capacidade de carga de até 5Kg. Possui três rodas de cada lado, sendo estas ligadas em paralelo, assim necessitando apenas de dois pontos de alimentação DC, uma para as rodas da direita e outra para as rodas da esquerda.

2.2 HARDWARE

O trabalho é composto basicamente por três placas:

1. Uma responsável por gerenciar as funcionalidades do ROS (*Panda Board*);

2. Um *Arduino Uno* para a interface entre o ROS e o controlador dos motores (*Wild Thumper Controller*) devido a limitação de memória deste controlador;
3. E o controlador dos motores do *Wild Thumper*.

2.2.1 Panda Board ES

A Panda Board ES é uma placa de desenvolvimento que utiliza o chip OMAP4460, baseado no ARM Cortex-A9, de produção da *Texas Instruments* (TEXAS, 2012). O objetivo da plataforma é fornecer uma placa de desenvolvimento inicial para pessoas que tem interesse em sistemas embarcados (figura 3).

Possui diversos componentes, como slot para cartão de memória SD ou MMC de até 32Gb, duas saídas HDMI, uma para saída de vídeo e outra para uma tela de expansão LCD, Wi-Fi, interface serial RS232, saídas de áudio, 2 portas USB (USB-A) e uma interface ethernet. Oficialmente suporta o sistema *Android* e GNU/Linux *Ubuntu*, o qual foi utilizado no desenvolvimento do trabalho.
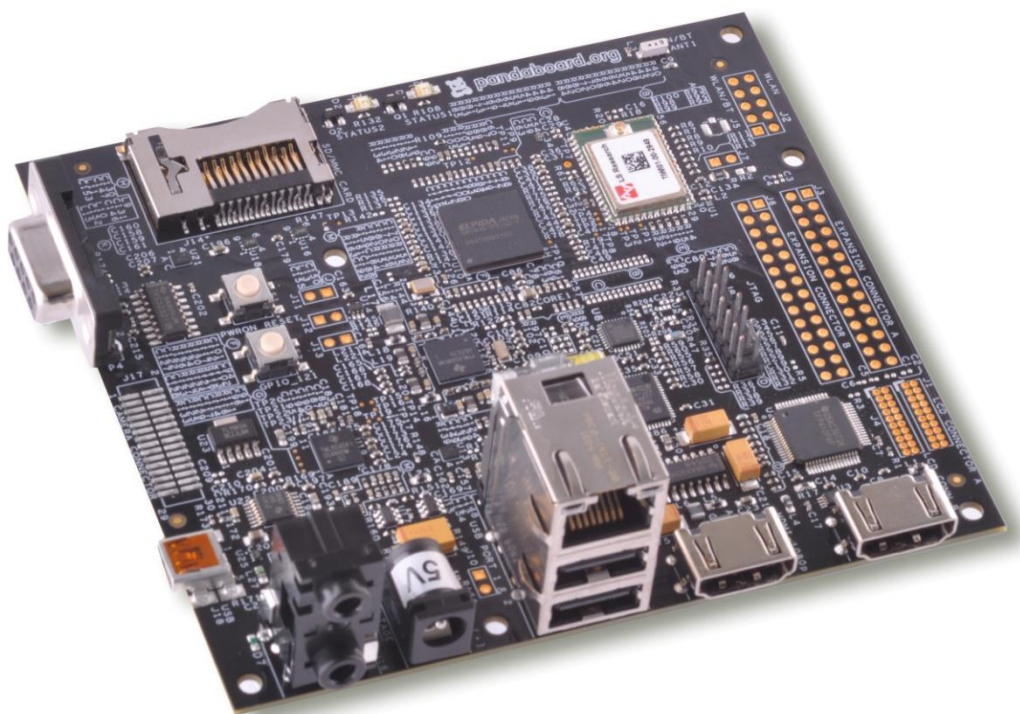
Figura 3 – Imagem da placa de desenvolvimento Panda Board
Fonte: PANDA (2012).

2.2.2 Placa de controle (*Wild Thumper Controller*)

A placa, que pode ser vista na figura 4, possui os componentes necessários para a utilização dos recursos do veículo, suportando até 15A por ponte H (circuito que permite o controle do sentido e da corrente que chega aos motores) também possuindo um sistema de recarga para alguns tipos de bateria. O modelo utiliza uma bateria padrão de veículos rádio controlados (7.2V ou 7.4V e 5A). Vem com um microcontrolador AVR (ATMega168), com o *bootloader* arduino, sendo que sua interface de comunicação pode ser realizada por USB ou serial (I2C, TTL).
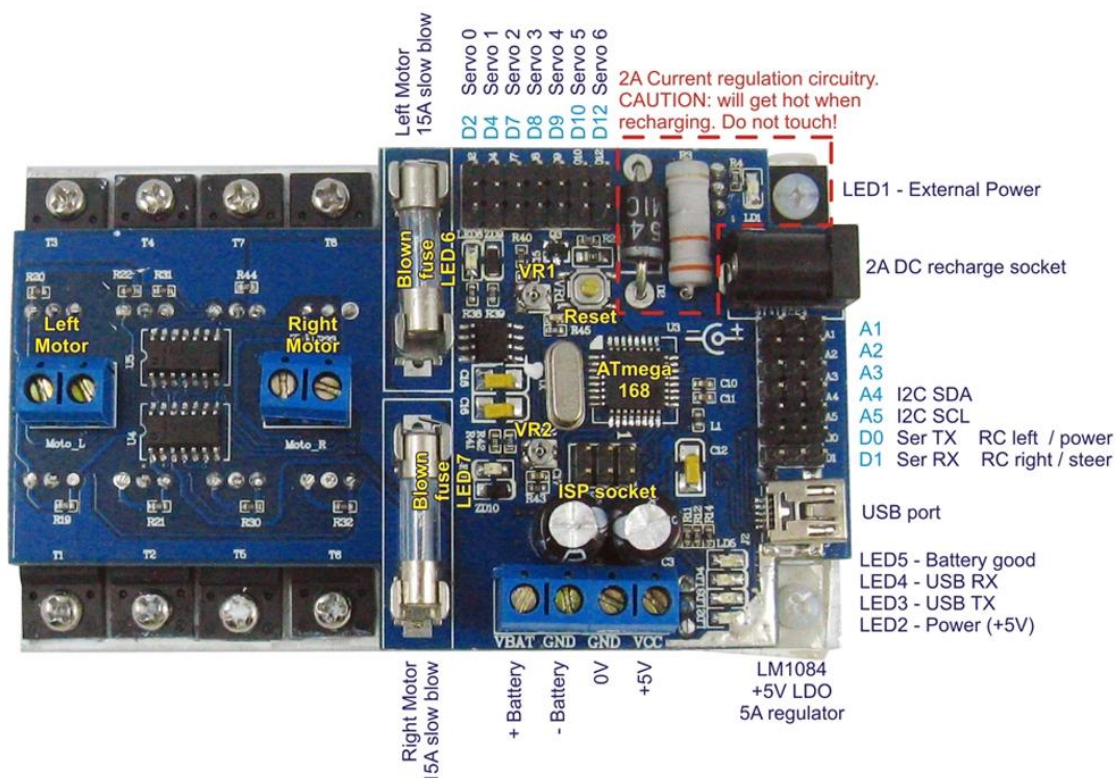
Figura 4 - Imagem da placa de controle para o Wild Thumper

Fonte: WILD BOARD (2012).

2.2.3 Arduino Uno

A plataforma Arduino disponibiliza placas de desenvolvimento de software e hardware abertos (*Open Source*), a qual provê uma interface fácil e flexível de eletrônica e programação para *hobbystas* e entusiastas na criação de objetos interativos. Sua linguagem de programação é chamada de *Arduino Programming Language*, baseada na linguagem de programação *Wiring*, que é um *framework* de programação para micro controladores (ARDUINO HOME, 2013).

Neste trabalho utiliza-se a placa Arduino UNO R3 (*revision 3*) (figura 5), a qual possui um microcontrolador AVR modelo *Atmega328*, como controlador, e um FTDI *USB-to-serial* como conversor usb/serial. Está placa é considerada a referência para as novas placas que são projetadas pelo grupo de desenvolvimento do *Arduino* (ARDUINO UNO, 2013).

Todos os sistemas baseados no *Arduino* possuem ao menos duas seções em seus programas:

- ***Setup***: configuração inicial de parâmetros para a placa, sendo que será executado uma única vez;
- ***Loop***: local onde serão implementadas as funcionalidades do programa; fica em execução contínua enquanto a placa estiver ligada.
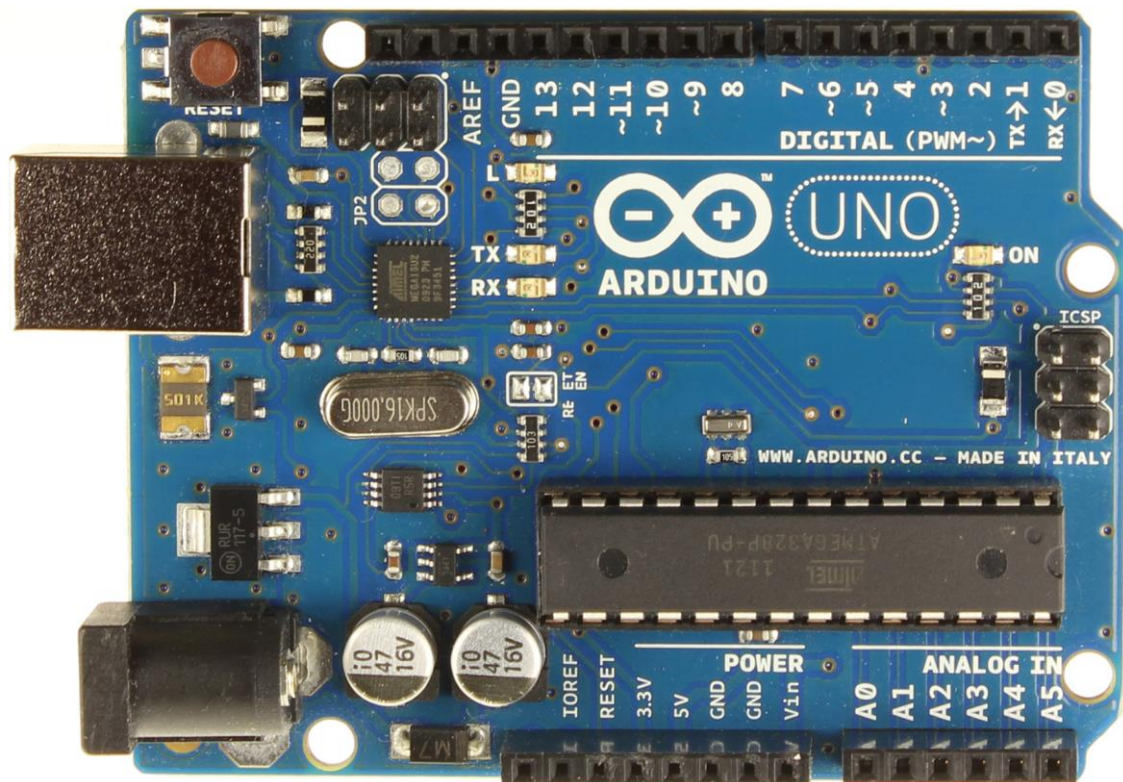
Figura 5- Arduino Uno R3

Fonte: UNO IMAGE (2013).

2.3 SOFTWARE

O sistema utilizará dois SO, um para o celular e outro para a placa no veículo, a qual realizará o papel de *node master* do ROS. Os dois sistemas serão baseados em Linux: *Android* (plataforma móvel) e Ubuntu (plataforma embarcada).

2.3.1 Android

O *Android* é um *software stack* que roda sobre o sistema operacional Linux. O software teve origem com a empresa *Android Inc.*, a qual foi adquirida pelo *Google Inc.*, em 2005. Nas mãos do Google o software sofreu uma refor-

mulação, passou a ter uma licença *Open Source* a *ASL 2.0* (*Apache Software Foundation*) e seu desenvolvimento foi delegado para o grupo de trabalho *Android Open Source Project* (AOSP), pertencente à organização *Open Handset Alliance* (OHA), liderado pelo Google (*ABOUT ANDROID, 2012*).

Atualmente o Google é responsável pela engenharia e desenvolvimento das novas versões do *Core Android*, enquanto as outras companhias pertencentes ao grupo são responsáveis por *bugfixes* e adaptações necessárias para as versões que já foram lançadas (*ABOUT ANDROID, 2012*).

O *Android*, sendo um software *stack*, fica responsável por prover as funcionalidades necessárias para que as aplicações possam executar sobre ela. As aplicações são desenvolvidas para a plataforma através da linguagem Java utilizando-se o *Android* SDK. O código de máquina gerado pelo SDK é interpretado pela máquina virtual Dalvik. Este nada mais é do que um interpretador otimizado de código de máquina Java, visando o melhor uso de recursos do celular (ex.: economia de energia).

Aplicações que necessitam de maior desempenho podem utilizar o *Native Development Kit* (NDK), com a qual as aplicações são desenvolvidas na linguagem de programação C/C++. A estrutura interna do *Android* pode ser visualizada através da figura 6.
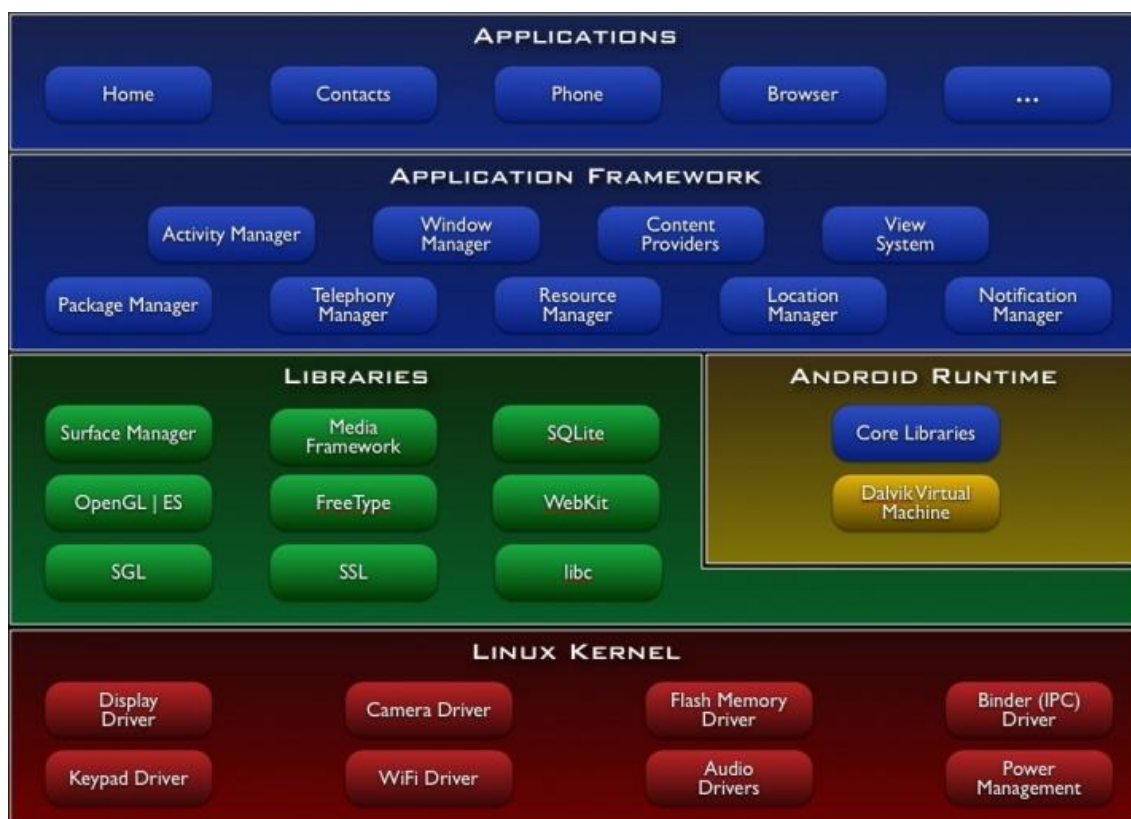
Figura 6 - Arquitetura de funcionamento do Android
Fonte: ANDROID WHAT IS (2012).

Todo o profissional tem total acesso às APIs utilizadas para o desenvolvimento do Core e pode utilizá-las caso julgue necessário. As aplicações são compostas por diversos componentes, considerados os blocos de montar do *Android*; são eles (ANDROID FUNDAMENTALS, 2012):

- ***Activities:*** representam as janelas do Android, por exemplo, uma aplicação de e-mail pode possuir uma janela para listar a caixa de entrada e outra para escrever novas mensagens, e cada uma delas é considerada uma *activity*.
- ***Services:*** rodam em *background* no dispositivo, não possuem nenhuma *activity*. Normalmente são utilizados em atividades de longa duração ou em algumas tarefas, como envio de dados através da rede, para não bloquear a interface do usuário.
- ***Content Provider:*** utilizados para que aplicações compartilhem uma informação, como, por exemplo, o número de algum contato no telefone. Os dados normalmente são armazenados em arqui-

vos, localmente ou através da *web*, utilizando o banco de dados SQL Lite.

- ***Broadcast Receivers:*** enviam mensagens de uma aplicação para todas as outras aplicações no sistema. Por exemplo, quando a tela do celular é desligada, envia-se uma mensagem de notificação, sendo que qualquer um pode utilizar esta informação.

A plataforma de desenvolvimento (SDK) oferece suporte para *Windows*, *Linux* ou *Mac,* desde que o *Java Development Kit* (JDK) esteja corretamente instalado. Em Android, o desenvolvedor pode trabalhar com diversas revisões do SDK ao mesmo tempo. O editor de código padrão para o Android é o Eclipse (uma plataforma padrão para desenvolver programas Java); seu uso não é obrigatório, mas é altamente recomendado pelas diversas facilidades disponíveis. A plataforma também oferece um emulador, o qual pode reproduzir a versão Android desejada.

2.3.2 Robot Operating System (ROS)

O projeto teve início com Morgan Quigley, em Stanford, quando criou o sistema *Switchyard*, na qual o ROS é baseado (ROS PLATFORM, 2012). Atualmente a principal mantenedora do projeto é a *Willow Garage*, um grupo que trabalha com a construção de aplicativos com código e hardware aberto para uso na robótica; outros projetos do grupo são o *Point Cloud Library* (PCL) e *Open Source Computer Vision* (OpenCV). O ROS pode ser instalado em qualquer plataforma compatível com sistemas Linux ou Unix (WILLOW, 2012).

O Robot OS é um *framework*, que vem sendo utilizado tanto no meio acadêmico como em projetos industriais de robótica. O principal fator é a facilidade gerada no desenvolvimento de softwares para este fim, viabilizando a reutilização de código e promovendo uma interface modular e distribuída.

A plataforma é composta fundamentalmente por três partes (ROS INTRODUCTION, 2012):

- **Nível de Sistema de Arquivos *(FileSystem Level):*** é o nível onde estão disponíveis os recursos no disco do sistema;

- **Nível Grafo de Computação *(Computation Graph Level):*** é o nível onde se estabelece a conexão *peer-to-peer* da rede de processos do ROS;

- **Nível de Comunidade *(Community Level):*** é o nível que permite a troca de informações entre diferentes grupos de pesquisa.

Novos módulos para o ROS podem ser desenvolvidos em diversas linguagens de programação, como C++, Python e Lisp.

2.3.2.1 File System Level

Seus componentes fundamentais são (ROS CONCEPTS, 2012):

- **Pacotes *(Packages)*:** compõe a unidade da organização de software utilizada no ROS; contêm processos, dependências, arquivos de configuração e qualquer coisa que possa ser útil para entendimento do pacote;
- **Pilhas *(Stacks)*:** coleções de *packages* que reúnem funcionalidades para o sistema, como, por exemplo, *stack* de navegação.
- **Arquivos de Manifesto *(Manifest files)*:** contêm informações sobre os *packages* e *stack*, como dependências entre eles, *flags* de compilação e licença.
- **Tipos de Mensagem *(Message types)*:** define a estrutura de dados da mensagem que será enviada pelo ROS.
- **Tipos de Serviço *(Service types)*:** define a estrutura que será utilizada para requisitar e responder serviços ROS.

2.3.2.2 Computation Graph Level

Neste nível são gerenciadas as conexões e é criado o grafo de interações. Os componentes fundamentais deste nível são (ROS CONCEPTS, 2012):

- **Nós *(Nodes)*:** são processos que controlam algum tipo de evento realizando processamento necessário para tal; em um robô podem-se ter diversos *nodes* rodando (ex.: um para calcular localização, um para medir distância).

- **Mestre *(Master)*:** funciona como um servidor de nomes (DNS), permitindo que diversos *nodes* localizem seus “parceiros”.
- **Parâmetros do Servidor *(Parameter Server)*:** permite que chaves sejam guardadas em uma localização central.
- **Mensagens *(Messages)*:** a comunicação entre os diversos nós é feita através de trocas de mensagens. As mensagens podem conter tanto os tipos de variáveis padrão em C (integer, float, char etc), como também estruturas aninhadas (como *structs* em C).
- **Tópicos *(Topics)*:** é o nome utilizado para identificar o conteúdo de uma mensagem. As mensagens são transportadas no formato de publicação/interessado (*publisher/subscriber*): um *node* publica uma mensagem, então o *node* interessado em um tópico coleta e processa a informação.
- **Serviços *(Services)*:** apesar de o modelo de publicação/interessado ser flexível, muitas vezes é necessário que seja utilizado o modelo tradicional de sistemas distribuídos, requisição/resposta (*request/response*), o qual é realizado através de *services*.
- **Bolsa *(Bags)*:** é uma forma de armazenar informações de dados coletados, como, por exemplo, de um sensor. São utilizados para desenvolver e testar as implementações de pacotes.

2.3.2.3 Community Level

Permite compartilhar módulos e informações entre comunidades de desenvolvimento espalhadas pelo mundo. A plataforma possui diversos canais, dentre eles, os principais são (ROS CONCEPTS, 2012):

- **Distribuições:** são coleções de versões de *stacks* que podem ser facilmente instalados.

- **Repositórios:** são mantidos em uma rede compartilhada e independente, visando melhorar a qualidade do serviço e a redução de custos.

- **ROS wiki:** é a maior fonte de informação sobre o ROS, contendo documentação de *packages*, *stacks* e tutoriais.

2.3.2.3 Funcionamento

O ROS possui um mestre (*Master*) que é responsável por armazenar informações de tópicos e serviços dos nós que a ele estão conectados. Quando um novo nó conecta-se ao mestre, este nó envia sua mensagem de registro, contendo informações dos tópicos e serviços que assina ou pública. Em contrapartida o mestre informa outros nós que tenham interesse nas informações geradas por este novo nó, criando conexões dinamicamente (ROS CONCEPTS, 2012).

O mestre funciona como um *DNS server*, ou seja, os nós se comunicam diretamente, e utilizam o serviço do mestre como *lookup server* para identificar nós compatíveis com as informações esperadas por ele. A conexão entre tópicos dos nós é feita através da associação de nome do tópico: um nó publica, outro ou outros nós assinam o tópico e recebem as mensagens respectivas. Isso associado com a renomeação de tópicos, permite que o sistema tenha várias mensagens de um mesmo tipo circulando no ambiente sem que haja conflito entre elas (ROS CONCEPTS, 2012). A figura 7 exemplifica as formas de troca de mensagens.
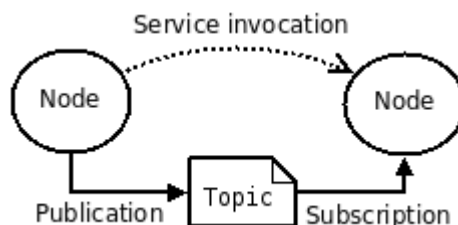
Figura 7 - Estrutura básica do processo de troca de mensagens entre nós ROS
Fonte: ROS BASIC (2013).

2.3.3 GNU/Linux Ubuntu

É o sistema operacional utilizado na plataforma embarcada do projeto. A versão utilizada é a 12.04.1“Precise Pangolin” Long Term Support, pois esta é a versão na qual o ROS “Groovy” é oficialmente suportado. O GNU/Linux Ubuntu é uma das distribuições *Unix Like* mais fáceis de se manusear, e por isso, uma das mais populares. Outra vantagem relacionada à distribuição é que esta possui a imagem necessária para as placas que possuem processadores OMAP série 4, o mesmo presente na *Panda Board*.

3 DESENVOLVIMENTO

Para que fosse viável o desenvolvimento do trabalho, abordou-se tópicos de maior impacto primeiramente. Como o ROS é o componente de maior importância, o primeiro passo foi fazer com que o mesmo estivesse operante na plataforma ARM, ou seja, na *Panda Board*.

3.1 PANDA BOARD

Para que o ROS pudesse ser executado na placa, alguns tópicos foram abordados previamente, como a própria instalação do sistema base para executar o *framework*. O sistema suportado oficialmente pelo ROS é o *Ubuntu 12.04 LTS*, portanto esta foi a plataforma escolhida.

O primeiro passo foi determinar a funcionalidade do SO executando na arquitetura ARM. Isso definiria a factibilidade do projeto: se o processo se mostrasse muito complexo ou lento, o trabalho não poderia ser facilmente aplicado de forma didática.

Com o SO executando de maneira aceitável, alguns parâmetros deveriam ser configurados, como a utilização da rede Wi-Fi, que determinaria a qualidade na recepção dos dados remotamente, ou seja, não poderia ocorrer uma grande quantidade de pacotes de rede perdidos. O parâmetro observado neste teste é o não comprometimento da funcionalidade do robô, ou seja, sua movimentação.

Tendo realizados os testes com o SO e a rede sem fio, o próximo passo foi determinar a usabilidade do *framework* ROS, verificando a complexidade de sua instalação e utilização. Em seguida, verificando a compatibilidade dos pacotes necessários com a arquitetura ARM, sendo eles:

- ***ros-groovy-ros-base***: responsável pela instalação dos pacotes base do sistema ROS;
- ***ros-groovy-rosserial-arduino***: responsável por gerar a biblioteca contendo formatos de mensagens e código necessário para que o *Arduino* negocie com a ponte criada pelo pacote *rosserial python* (cliente);
- ***ros-groovy-rosserial-python***: responsável por realizar e gerenciar a interface entre a rede TCP/IP criada pelo ROS e a conexão serial proveniente do *Arduino*.

Neste passo buscou-se verificar apenas a disponibilidade dos pacotes binários do *rosserial* para a arquitetura ARM, sem que o teste de funcionalidade fosse realizado. A estrutura de nós do ROS, com o *core* da aplicação em destaque, pode ser observada na figura 7.
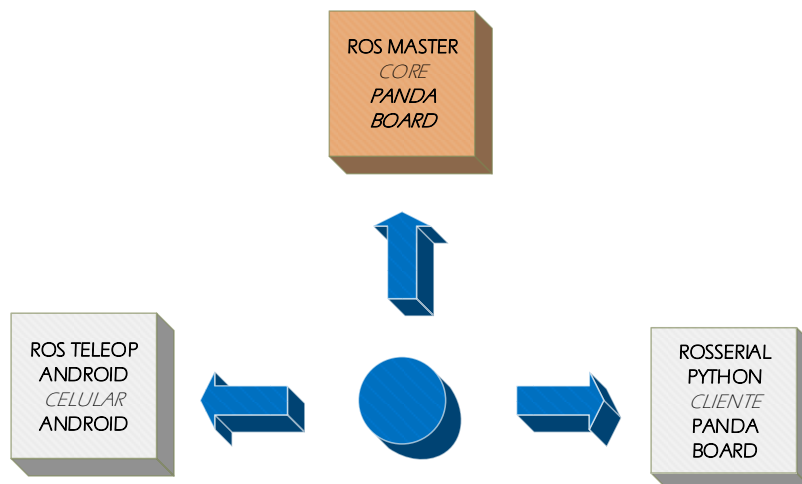
Figura 8 - ROS core

Fonte: Autoria Própria.

3.2 SMARTPHONE ANDROID

Após concluir as atividades iniciais com a *Panda Board*, o passo seguinte foi verificar a aplicabilidade do programa de teleoperação do ROS disponível para *Android*, tópico em destaque na figura 9. Ele está disponível através da *Play Store*, do Google (PLAY STORE, 2013), e com isso a instalação foi simples, como pode ser visualizado na figura 10.

Para verificar seu funcionamento, bastou entrar com um URI de um ROS *core* válido, o que pode ser aferido através da utilização de máquinas virtuais com o ROS em execução, assim podendo listar os tópicos e nós ativos. Com o *core* acessível para a aplicação, pode-se realizar testes de funcionamento da aplicação através da utilização de um simulador disponível nos pacotes do ROS, o *turtlesim* (TURTLESIM, 2013), que consiste de uma tela com uma tartaruga, a qual recebe mensagens para tele operação e movimenta a tartaruga graficamente.
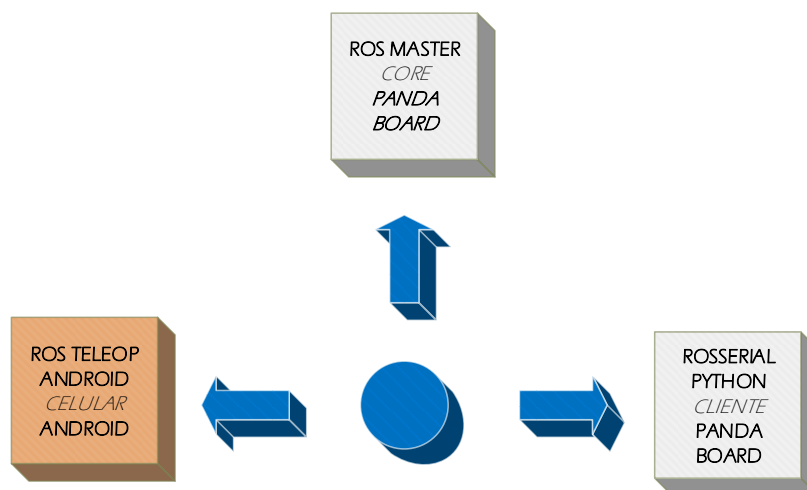
Figura 9 - Nó ROS de teleoperação, executando em um Android

Fonte: Autoria Própria.

Verificando a compatibilidade dos movimentos com os sinais de controle, pode-se determinar a viabilidade da aplicação na geração dos comandos para a plataforma desejada, ou seja, no *Wild Thumper*. Tendo a aplicação em funcionamento e gerando sinais compatíveis com os desejados para o robô, pode-se dar início à coleta das mensagens geradas, e assim, analisar e aplicar os resultados no desenvolvimento da aplicação executada no *Arduino*.
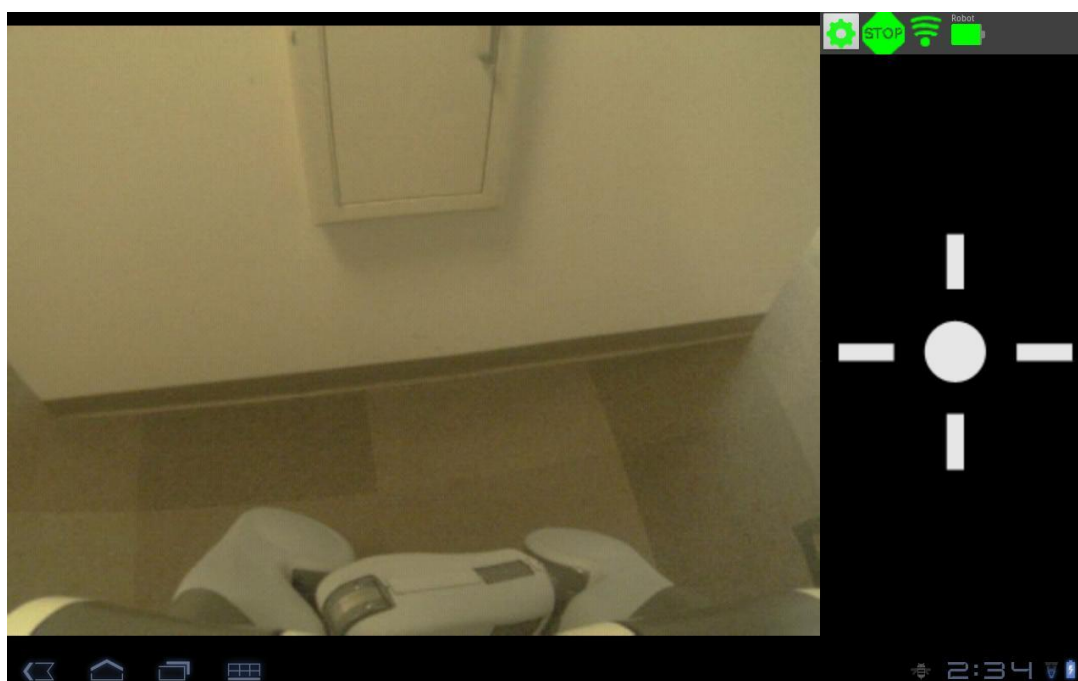
Figura 10 – Snapshot da aplicação Android com câmera integrada, retirado da internet

Fonte: PLAY TELEOP (2013).

3.3 ARDUINO

Este foi o último passo em que o desenvolvimento se deu pontualmente, sem que houvesse a integração entre os demais componentes do projeto. Para isso, foi fundamental determinar quantas placas seriam necessárias. A princípio apenas uma placa seria necessária, sendo esta o *Wild Thumper Controller*, mesmo com as várias limitações de hardware. Para verificar a viabilidade do uso deste hardware, foi adicionado o *stack* do ROS ao ambiente de desenvolvimento do *Arduino* e então compilar os programas exemplos disponibilizados pelo site do ROS (ROSSERIAL TUTO, 2013). Com isso, verificou-se uma das limitações de hardware, o tamanho máximo de um programa na memória do controlador (14 kB) foi atingido. Optou-se então por trabalhar com duas placas, uma um pouco mais potente para realizar a interface com o ROS (*Arduino Uno*) e outra para controle dos motores (*Wild Thumper Controller*). Essa interação pode ser visualizada na figura 11.
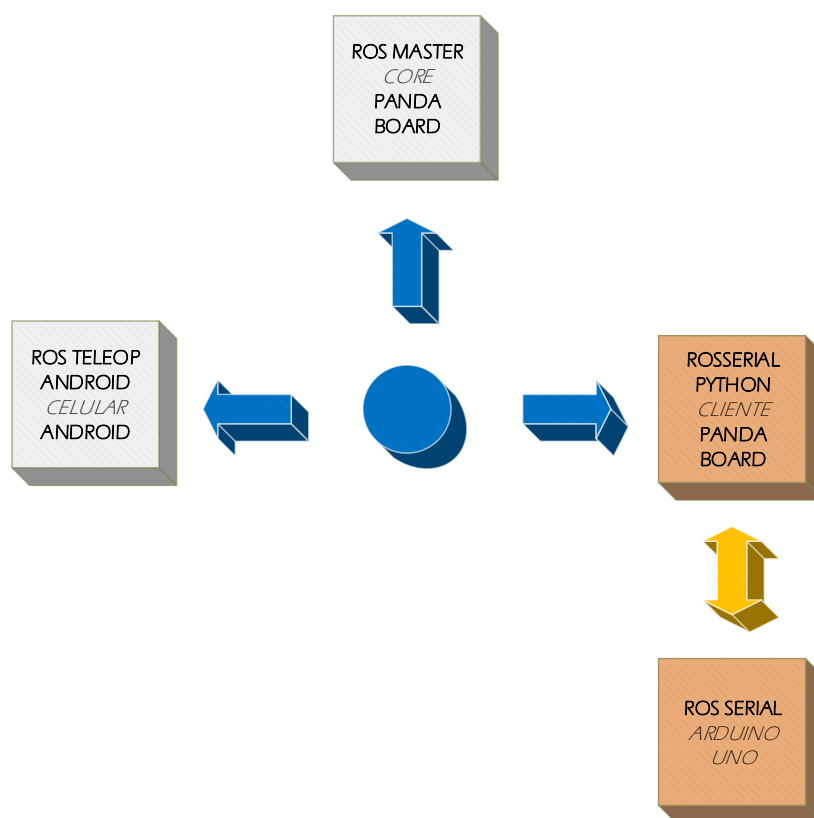
Figura 11 - Nodo ROS que realiza o papel de "gateway" para placa Arduino Uno

Fonte: Autoria Própria.

3.3.1 Arduino Uno

Esta placa foi incluída no projeto para suprir as limitações de memória existentes na placa que realiza o controle dos motores. Esta foi a abordagem encontrada mais simples, sem que códigos fonte precisassem ser alterados. O teste para verificar se a placa poderia ser utilizada com o ROS foi similar ao teste com o controlador dos motores, utilizando os programas exemplos da stack do ROS para o *Arduino* e observando se o mesmo conseguiria estabelecer uma conexão estável com o *core* e abrigar a aplicação como um todo. A figura 12 exibe a aplicação do *Arduino Uno* no projeto, realizando a interface entre o controlador dos motores e o ROS.

Figura 12 - Arduino Uno em destaque por interface realizada entre ROS e controlador dos motores

Fonte: Autoria Própria.

Após averiguar a funcionalidade utilizando formatos básicos de mensagem, optou-se por testar o formato de mensagem utilizado no projeto. Esta mensagem é chamada pelo ROS de *Twist*, e expressa os valores de velocidade em sua parte linear e angular, ou seja, dois conjuntos de dados, cada um com três valores, X, Y e Z. A mensagem gerada pela aplicação *Android* tem os valores de X linear e Z angular preenchidos.

Como os valores de Z angular aplicados não poderiam ser corretamente aferidos, pois os motores disponíveis no *Wild Thumper* não possuem encoder, o valor foi utilizado para determinar se o veículo deveria seguir para direita ou esquerda, e os valores de X linear para determinar a velocidade para frente ou para trás.

Possuindo estes dados como base, pode-se então determinar o funcionamento da aplicação.

3.3.2 Wild Thumper Controller

Figura 13 - Controlador dos motores que recebe dados através da conexão I2C
Fonte: Autoria Própria.

Com a placa de interface com o ROS em funcionamento, ainda precisava-se de um método de comunicação entre ela e o controlador dos motores, e para isso optou-se pela utilização da comunicação I2C, pois com a utilização da biblioteca *Wire* (ARDUINO WIRE, 2013) disponibilizadas pelo *Arduino*, a interface poderia ser realizada de forma simples. Outra vantagem da biblioteca é que ela pode ser utilizada com programação orientada a eventos, o que facilita a integração entre as placas. A figura 13 representa a ligação I2C entre o *Arduino Uno* e o controlador dos motores, com destaque para o último.

Para verificar se os valores de velocidade, direção e sentido estavam sendo corretamente processados pelo *Arduino Uno*, foi necessária uma ferramenta auxiliar, disponível na interface da ferramenta de desenvolvimento do *Arduino*, de nome *serial monitor*, capaz de realizar leituras da porta serial e então mostrá-las para o usuário. Neste passo, os tópicos no formato de mensagem *Twist* foram gerados com auxílio do comando *rostopic* do ROS, o qual permite a publicação de tópicos via linha de comando; os dados são recebidos pelo *Arduino Uno*, processados para o formato velocidade, direção e sentido, e então enviados para o controlador dos motores, o qual exibe os valores na ferramenta *serial monitor*.

Com os dados sendo processados corretamente, pode-se integrar os três nós do ROS e verificar seu funcionamento. Assim que os processos funcionavam corretamente, pode-se ligar os motores do veículo e então regular o *Wild Thumper Controller* para que o movimento do veículo fosse uniforme. O esquema elétrico da ligação das placas pode ser observado na figura 14.
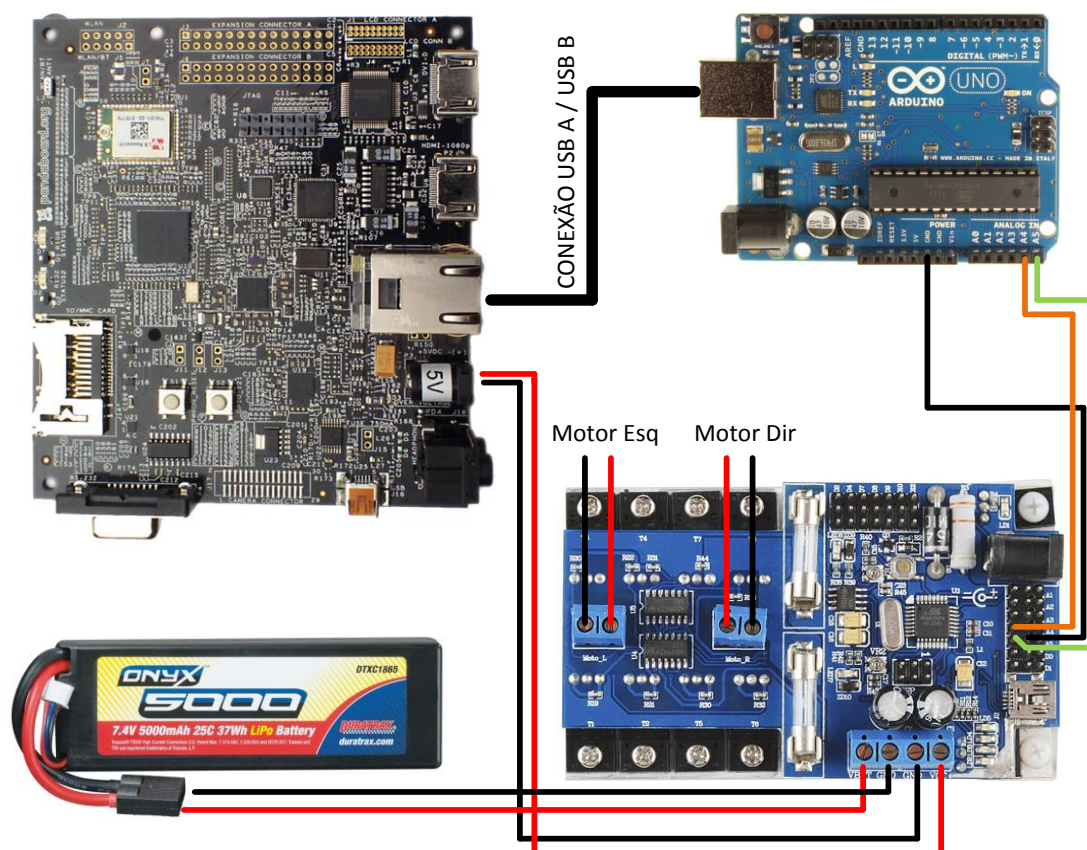
Figura 14 - Integração elétrica final do trabalho

Fonte: Autoria Própria.

3.4 TUTORIAIS

Com o desenvolvimento prático do trabalho concluído, pode-se iniciar o trabalho nos materiais de aprendizado. Assim o material foi escrito seguindo os mesmos passos do desenvolvimento do trabalho prático em si, porém sem que os testes intermediários fossem descritos. Os tutoriais, que estes podem ser encontrados no apêndice D, são:

- Instalação básica do sistema Ubuntu 12.04;
- Configuração básica com a ativação da rede Wi-Fi na Panda Board;
- Instalação do framework ROS em SO Ubuntu para arquitetura ARM;
- Esquema e montagem elétrica dos equipamentos no *Wild Thumper*;
- Manual com comandos e dicas básicas do ROS;
- Como executar a aplicação do trabalho.

4 RESULTADOS

Durante o desenvolvimento do trabalho alguns testes foram realizados visando abordagens que trouxessem melhores resultados, facilitando o aprendizado do ROS e retirando o foco de possíveis problemas encontrados em outros locais, como SO Ubuntu e aplicativo *Android*.

4.1 TESTES COM PANDA BOARD

Inicialmente o foco do trabalho foi na instalação e manutenção do SO instalado na *Panda Board*. O sistema utilizado foi o *Ubuntu*, que a princípio mostrou-se instável com atualizações que corrompiam o sistema de arquivos, impossibilitando o boot. Para solucionar este problema alternativas foram exploradas, como a utilização de outras distribuições Linux como o *OpenSuSE* para ARM, que embora tenha se mostrado muito estável, não possuía suporte oficial da comunidade do ROS, o que acarretou em incompatibilidade com algumas das bibliotecas que foram utilizadas e por isso seu uso foi vetado.

Assim o foco do projeto voltou a ser o Ubuntu, que após o lançamento dos pacotes binários do ROS para a plataforma ARM tornou sua instalação simples, bastando apenas utilizar uma versão de SO compatível com o repositório e, com isso, todos os pacotes necessários puderam ser instalados com apenas uma linha de comando.

4.2 TESTES COM APLICAÇÃO ANDROID

Após comprovar o funcionamento do ROS no SO da *Panda Board*, os testes seguintes com o ROS foram realizados com o auxílio de máquinas virtuais. O propósito dos testes foi averiguar o funcionamento da aplicação de tele operação. Para isso utilizou-se um *smartphone* com SO *Android* e instalou-se a aplicação desejada, ROS *Teleop* da Willow Garage. Apesar de sua configuração ser relativamente simples, bastando configurar o endereço e porta em que o ROS *core* está executando, a aplicação se mostrou muito instável, muitas vezes acarretando em seu travamento.

Como alternativa a esta aplicação pouco estável, foi iniciado o desenvolvimento de uma aplicação WEB para verificar a sua viabilidade, mas novamente os resultados não foram muito animadores, pois a interface disponibilizada pelos navegadores do *Android* não consegue atender os requisitos de interface

necessários para o controle do veículo, aceitando apenas um toque na interface e muitas vezes exibindo o menu de contexto.

Isso tornou-se o fator decisivo na escolha pela aplicação original, disponibilizada pela Willow Garage. Apesar de ter um funcionamento duvidoso, atende aos requisitos necessários, sendo que os únicos parâmetros que necessitaram de configuração no ROS *core* para que a aplicação pudesse gerar os tópicos de tele operação, foram os chamados *robot/name*, para o nome que é exibido no display do smartphone, e *robot/type*, para a identificação do formato em que o tópico deve ser publicado; neste caso utilizou-se o robô *turtlebot* como base.

4.3 TESTES COM ARDUINO

Com boa parte do trabalho já encaminhado, iniciaram-se os testes com o pacote *rosserial*, utilizado para realizar a interface entre a rede TCP/IP criada pelo ROS e a conexão serial utilizada pelo *Arduino*. Nas origens do projeto, apenas o controlador dos motores se fazia necessário, pois este já tinha uma plataforma compatível com o *Arduino* e assim não necessitaria de mais nenhum controlador externo, como o *Arduino Uno*. Porém com o andamento dos testes duas limitações de hardware foram encontradas: a limitação de memória RAM (1kB), o que acarretava em perda de sincronia com o nodo *rosserial_python* (cliente), utilizado para a interface com a placa; e o fato da inclusão da biblioteca do ROS (*rosserial_arduino*) no programa desenvolvido para o controlador fazer com que o tamanho da aplicação binária extrapolasse o tamanho máximo permitido pelo controlador, limitado a 14kB.

Como alternativa, adicionou-se uma placa *Arduino Uno* para realizar a interface entre o ROS e o controlador dos motores. Com isso a lógica necessária para o desenvolvimento da aplicação tornou-se muito mais simples, fazendo com que o controlador dos motores apenas recebesse os parâmetros de velocidade, direção e sentido, enquanto o *Arduino Uno* realiza o processamento dos dados recebidos através da interface com o ROS e os envia utilizando uma conexão I2C. Para averiguar o funcionamento correto no processamento das mensagens *Twist*, tópicos compatíveis com este formato foram criados através do comando *rostopic* e os valores processados foram verificados através da leitura da conexão USB/Serial ainda disponível no controlador dos motores

(*Wild Thumper Controller*). Os códigos utilizados para o mestre e para o escravo do I2C estão disponíveis nos apêndices B e C, respectivamente.

4.4 TESTE DA SOLUÇÃO COMPLETA

Após concluídos os testes isolados, integrou-se a Panda Board e as placas do Arduino. O procedimento utilizado foi utilizar o rosserial python da Panda Board para conectá-la com o Arduino Uno, e verificar se a placa conseguiria conectar-se ao sistema. Com os testes entre a Panda Board e o Arduino Uno concluídos, integrou-se o controlador dos motores ao sistema e mensagens do formato *Twist* foram geradas por linha de comando, verificando-se os valores que eram recebidos no controlador dos motores. Após isso, integrou-se o celular *Android*, para geração das mensagens de controle do veículo (*Twist*).

Todos os componentes foram integrados sem maiores problemas e, a partir daí, iniciou-se o desenvolvimento dos tutoriais. Estes se basearam no processo final, ou seja, sem que passos intermediários fossem descritos. O objetivo destes é esclarecer tópicos muitas vezes obscuros a iniciantes na plataforma do ROS. A figura 15 demonstra a solução completa.
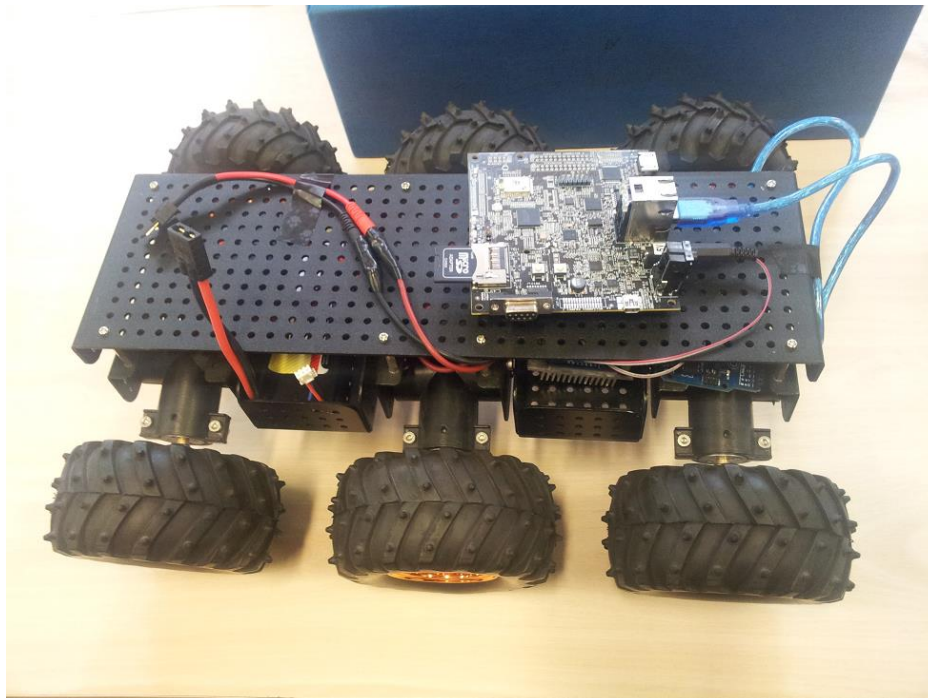
Figura 15 - Solução completa

Fonte: Autoria Própria.

5 CONSIDERAÇÕES FINAIS

O ROS como ferramenta de trabalho se mostrou muito útil, pois permite, com facilidade, integrar diversas funcionalidades a um robô. Isso se torna muito útil quando o objetivo do projeto é desenvolver um algoritmo específico, como por exemplo, um algoritmo de mapeamento de ambiente, ou de navegação. Para isso bastaria usar um robô compatível com o ROS, recebendo sem maiores problemas quase todas as funcionalidades desejadas, e então trabalhar quase que exclusivamente no algoritmo desejado.

Porém isso não se dá sem dificuldades, durante o desenvolvimento do trabalho vários pacotes se mostraram instáveis e por muitas vezes incompatíveis com a versão do sistema, no caso *Groovy*. Como o ROS ainda é considerada uma plataforma nova, sendo oficialmente disponibilizada em 2010, estes problemas tendem a se estabilizar quando o *core* do *framework* estiver mais maduro.

Outra dificuldade encontrada foi a quantidade de conhecimentos necessários para se fazer bom uso do sistema, sendo eles de programação, de sistemas operacionais baseados no POSIX, de redes, de robótica e até de eletrônica. Os tutoriais e materiais didáticos disponíveis no site do ROS também vêm sendo trabalhados, de forma a facilitar o aprendizado do *framework*.

Adquirindo os conhecimentos necessários, o trabalho pode ser facilmente reproduzido e incrementado, podendo-se utilizá-lo como base para o desenvolvimento de um robô para estudo de navegação autônoma, apenas com a adição de alguns sensores e câmeras.

REFERÊNCIAS

ABOUT ANDROID, maio 2012. **Sobre o Android**. Disponível em: <**http://source.android.com/about/index.html**>. Último acesso em: 02 de julho de 2012.

ANDROID FUNDAMENTALS, maio 2012. **Fundamentos Android**. Disponível em: <**http://developer.android.com/guide/topics/fundamentals.html**>. Último acesso em: 02 de julho de 2012.

ANDROID WHAT IS, maio 2012. **O que é Android**. Disponível em: <**http://developer.android.com/guide/basics/what-is-android.html**>. Último acesso em: 02 de julho de 2012.

ARDUINO HOME, setembro de 2013. **Página oficial Arduino**. Disponível em: <**http://arduino.cc/**>. Último acesso em: 16 de setembro de 2013.

ARDUINO UNO, setembro de 2013. **Informações Arduino Uno**. Disponível em: < **http://arduino.cc/en/Main/ArduinoBoardUno**>. Último acesso em: 16 de setembro de 2013.

ARDUINO WIRE, setembro de 2013. **Biblioteca Arduino Wire**. Disponível em: <**http://arduino.cc/en/reference/wire**>. Último acesso em: 16 de setembro de 2013.

CHITTA, Sachin et al. **Planning for Autonomous Door Opening with a Mobile Manipulator**. International Conference on Robotics and Automation, 2010.

CHITTA, Sachin et al. **Tactile Object Class and Internal State Recognition for Mobile Manipulation**. International Conference on Robotics and Automation, 2010.

PANDA, dezembro de 2012. Informações Panda Board. Disponível em: <**http://theallbox.com/archive/2010/10/pandaboard-on-sale-now-for-174/**>. Último acesso em: 11 de dezembro de 2012.

PLAY TELEOP, setembro de 2013. **Página Teleop na Play Store**. Disponível em: **<https://lh4.ggpht.com/63VgIrSC4S-do6x4bYVIZAjNet1gWvImGYXI6LdGKhF61NMLMaSHEDPpFO9xHgk8tA=h900>**. Último acesso em: 16 de setembro de 2013.

PLAY STORE, setembro de 2013. **Loja de aplicativos Google**. Disponível em: <**https://play.google.com**>. Último acesso em: 16 de setembro de 2013.

ROBOCUP, setembro de 2013. **RoboCup oficial**. Disponível em: <**http://www.robocup.org/**>. Último acesso em: 16 de setembro de 2013.

ROS BASIC, outubro de 2013. **Básico ROS**. Disponível em: <**http://ros.org/images/wiki/ROS_basic_concepts.png**>. Último acesso em: 11 de setembro de 2013.

ROS CONCEPTS, dezembro de 2012. **Conceitos ROS**. Disponível em: <**http://www.ros.org/wiki/ROS/Concepts**>. Último acesso em: 11 de dezembro de 2012.

ROS COURSES, novembro de 2012. **Cursos com ROS**. Disponível em: <**http://www.ros.org/wiki/Courses**>. Último acesso em: 11 de dezembro de 2012.

ROS HECTOR, setembro de 2013. **Pacotes ROS Team Hector**. Disponível em: <**http://wiki.ros.org/tu-darmstadt-ros-pkg**>. Último acesso em: 16 de setembro de 2013.

ROS INDUSTRIAL, setembro de 2013. **ROS Industrial**. Disponível em: <**http://rosindustrial.org/**>. Último acesso em: 16 de setembro de 2013.

ROS INTRODUCTION, outubro de 2012. **Introdução ROS**. Disponível em: <**http://www.ros.org/wiki/ROS/Introduction**>. Último acesso em: 22 de outubro de 2012.

ROS LIST, novembro de 2012. **Lista de pacotes ROS**. Disponível em: <**http://www.ros.org/browse/list.php**>. Último acesso em: 11 de novembro de 2012.

ROS PAPERS, novembro de 2012. **Lista de artigos ROS**. Disponível em: <**http://www.ros.org/wiki/Papers**>. Último acesso em: 10 de dezembro de 2012.

ROS PLATAFORM, dezembro de 2012. **Plataforma ROS**. Disponível em: <**http://www.willowgarage.com/pages/software/ros-platform**>. Último acesso em: 14 de novembro de 2012.

ROSSERIAL TUTO, setembro de 2013. **Rosserial tutoriais**. Disponível em: <**http://wiki.ros.org/rosserial_arduino/Tutorials**>. Último acesso em: 16 de setembro de 2013.

TEAM HECTOR, setembro de 2013. **Página Team Hector**. Disponível em: <**http://www.gkmm.tu-darmstadt.de/rescue/**>. Último acesso em: 16 de setembro de 2013.

TEXAS, dezembro de 2012. **Informações OMAP4**. Disponível em: <**http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12843&contentId=53243**>. Último acesso em: 09 de dezembro de 2012.

TURTLE ROS, setembro de 2013. **Turtlebot Robot**. Disponível em: <**http://wiki.ros.org/Robots/TurtleBot**>. Último acesso em: 16 de setembro de 2013.

TURTLESIM, setembro de 2013. **Turtlesim Simulador**. Disponível em: <***http://wiki.ros.org/turtlesim***>. Último acesso em: 16 de setembro de 2013.

UNO IMAGE, setembro de 2013. **Arduino Uno Imagem**. Disponível em: <**http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpg**>. Último acesso em: 16 de setembro de 2013.

WILD BOARD, dezembro de 2012. **Wild Thumper Controller Sparkfun**. Disponível em: <**https://www.sparkfun.com/products/11057**>. Último acesso em: 10 de dezembro de 2012.

WILD THUMPER SPARK, outubro 2012. **Wild Thumper Sparkfun**. Disponível em: <**https://www.sparkfun.com/products/11056**>. Último acesso em: 24 de novembro de 2012.

WILLOW, dezembro de 2012. **Willow Garage About Us**. Disponível em: <**http://www.willowgarage.com/pages/about-us**>. Último acesso em: 14 de novembro de 2012.

WILLOW TELEOP, setembro de 2013. **Aplicação Android Teleop**. Disponível em: <**https://play.google.com/store/apps/details?id=ros.android.teleop&hl=pt_BR**>. Último acesso em: 16 de setembro de 2013.

APÊNDICE A – Orçamento

ORÇAMENTO			
Itens	Quantidade	Custo Unitário	Custo do Item
Wild Thumper	1 unidade	R$ 1.351,01	R$ 1.351,01
Placa controladora Wild Thumper	1 unidade	R$ 350,00	R$ 350,00
Panda Board	1 unidade	R$ 950,00	R$ 950,00
Arduino Uno	1 unidade	R$ 60,00	R$ 60,00
Bateria 7.2V 5A + Carregador	1 unidade	R$ 502,41	R$ 502,41
Celular Android	1 unidade	R$ 1.369,00	R$ 1.369,00
Cabos de alimentação	1 m	R$ 5,00	R$ 5,00
Roteador Wi-Fi	1 unidade	R$ 59,90	R$ 59,90
Gravador cartão SD	1 unidade	R$ 35,00	R$ 35,00
Conversor USB-Serial	1 unidade	R$ 32,00	R$ 32,00
Cabo USB-A / USB-B	1 unidade	R$ 5,00	R$ 5,00
Cabo USB-miniUSB	1 unidade	R$ 11,90	R$ 11,90
Cartão SD 16GB	1 unidade	R$ 44,50	R$ 44,50
Pacote papel A4 500 folhas	1 unidades	R$ 15,00	R$ 30,00
Cartucho de Impressão	1 cartucho	R$ 120,00	R$ 120,00
Horas de trabalho	200 h	R$ 8,00	R$ 1.600
		TOTAL	R$ 6.525,72

APÊNDICE B – Código *Slave* I2C (*Wild Thumper Controller*)

```
/*
 * Programa de controle do Wild Thumper através de I2C
 * Author: Guilherme Z.
 */
#include <ArduinoHardware.h>
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <Wire.h>

// Inicialização do nodo do ROS
ros::NodeHandle nh;
// Variáveis de aceleração e rotação
byte Leftmode;
byte Rightmode;
byte velPWM;

// Transmissão de dados para Slave
void sendData() {
    Wire.beginTransmission(2);
    Wire.write(Leftmode);
    Wire.write(Rightmode);
    Wire.write(velPWM);
    Wire.endTransmission();
}

void processaMensagem(const geometry_msgs::Twist& msg) {
    float fx, fz;
    int x,z;

    fx = msg.linear.x * 100;
    fz = msg.angular.z * 100;
    x = abs(fx);
    z = abs(fz);
    // verifica sentido de giro
    // direita, esquerda, frente, traz
    if (x < z) {
        velPWM = min(z, 255);

        // calcula curva
        if (msg.angular.z > 0) {
            Leftmode = 0;
            Rightmode = 2;
        } else {
            Leftmode = 2;
            Rightmode = 0;
        }
```

```
    } else if (x > z) {
        velPWM = min(x, 255);

        // calcula sentido
        if (msg.linear.x > 0) {
            Leftmode = Rightmode = 2;
        } else {
            Leftmode = Rightmode = 0;
        }
    } else {
        velPWM = 0;
        Leftmode = Rightmode = 1;
    }
}

// Função de Callback inicializada
void messageCb(const geometry_msgs::Twist& msg) {
    processaMensagem(msg);
    sendData();
}

// Inicialização subscribe do tópico de interesse
ros::Subscriber<geometry_msgs::Twist>         sub("WildThumper/teleop",
&messageCb);

// Configuração de Inicialização
//   - Inicializa ROS e tópico
//   - Inicializa Nodo Master para I2C
void setup() {
    pinMode(13, OUTPUT);
    nh.initNode();
    nh.subscribe(sub);
    Wire.begin();
}

// Loop Principal
//   - Receber dados via ROS
//   - Transmitir para slave dados importantes
void loop() {
    nh.spinOnce();
    delay(100);
}
```

APÊNDICE C – Código *Slave* I2C (*Wild Thumper Controller*)

ARQUIVO SLAVE.INO

```
/*
 * Programa de atuação nos motores do Wild Thumper
 * Author: Guilherme Z.
 */

// Headers Sistema
#include <Wire.h>
// Headers Locais
#include "IOpins.h"
#include "Constants.h"

// Variaveis de controle de Bateria
byte Charged = 1;    // 0=Descarregada  1=Carregada
unsigned int Volts;
unsigned int LeftAmps;
unsigned int RightAmps;
unsigned long leftoverload;
unsigned long rightoverload;
// Variaveis de controle de direção e velocidade do carrinho
byte Leftmode = 1;   // 0=reverse, 1=brake, 2=forward
byte Rightmode = 1;  // 0=reverse, 1=brake, 2=forward
byte velPWM;         // PWM value for motors speed / brake

void setup() {
  Wire.begin(2);                 // Junta-se ao barramento I2C com id 2
  Wire.onReceive(receiveEvent); // Registra evento
}

void loop() {
  checkBattery();
  delay(100);
}

// Função é executada sempre que um dado for recebido
void receiveEvent(int howMany) {
  Leftmode = Wire.read();          // receive byte as an integer
  Rightmode = Wire.read();         // receive byte as an integer
  velPWM = Wire.read();            // receive byte as an integer

  if (Charged) {
    moveCarro(); // processa mensagem
  } else {
    // Desliga motores caso a bateria esteja descarregada
    analogWrite(LmotorA, 0);
    analogWrite(LmotorB, 0);
```

```
    analogWrite(RmotorB, 0);
    analogWrite(RmotorA, 0);
  }
}

void checkBattery() {
  Volts = analogRead(Battery);     // read the battery voltage
  LeftAmps = analogRead(LmotorC);  // read left motor current draw
  RightAmps = analogRead(RmotorC); // read right motor current draw

  // is motor current draw exceeding safe limit
  if (LeftAmps > Leftmaxamps) {
    analogWrite(LmotorA, 0);       // turn off motors
    analogWrite(LmotorB, 0);       // turn off motors
    leftoverload = millis();       // record time of overload
  }

  // is motor current draw exceeding safe limit
  if (RightAmps > Rightmaxamps) {
    analogWrite(RmotorA, 0);       // turn off motors
    analogWrite(RmotorB, 0);       // turn off motors
    rightoverload = millis();      // record time of overload
  }

  // speed controller shuts down until battery is recharged
  // This is a safety feature to prevent malfunction at low voltages!!
  // check condition of the battery
  if ((Volts < lowvolt) && (Charged == 1)) {
    Charged = 0;  // bateria descarregada
    digitalWrite(13, HIGH-digitalRead(13));
  }
}

// Função de movimentação
void moveCarro() {
  if ((millis()-leftoverload)>overloadtime)
  {
    switch (Leftmode)
    {
      case 2:
      analogWrite(LmotorA,0);
      analogWrite(LmotorB,velPWM);
      break;

      case 1:
      analogWrite(LmotorA,velPWM);
      analogWrite(LmotorB,velPWM);
      break;
```

```
      case 0:
      analogWrite(LmotorA,velPWM);
      analogWrite(LmotorB,0);
      break;
    }
  }
  if ((millis()-rightoverload)>overloadtime)
  {
    switch (Rightmode)
    {
      case 2:
      analogWrite(RmotorA,0);
      analogWrite(RmotorB,velPWM);
      break;

      case 1:
      analogWrite(RmotorA,velPWM);
      analogWrite(RmotorB,velPWM);
      break;

      case 0:
      analogWrite(RmotorA,velPWM);
      analogWrite(RmotorB,0);
      break;
    }
  }
}
```

ARQUIVO IOpins.h

```
#define LmotorA 3  // Left  motor H bridge, input A
#define LmotorB 11 // Left  motor H bridge, input B
#define RmotorA 5  // Right motor H bridge, input A
#define RmotorB 6  // Right motor H bridge, input B

#define RCleft  0  // Digital input 0
#define RCright 1  // Digital input 1

#define S0      2  // Servo output 00
#define S1      4  // Servo output 01
#define S2      7  // Servo output 02
#define S3      8  // Servo output 03
#define S4      9  // Servo output 04
#define S5      10 // Servo output 05
#define S6      12 // Servo output 06

#define Battery 0  // Analog input 00
#define RmotorC 6  // Analog input 06
#define LmotorC 7  // Analog input 07
```

```
#define Charger 13 // Low=ON High=OFF
```

ARQUIVO Constants.h

```
//== MODE OF COMMUNICATIONS ==
#define Cmode 1 // Sets communication mode: 0=RC    1=Serial    2=I2C
#define Brate 115200 // Baud rate for serial communications

//== RC MODE OPTIONS ==
#define Mix 1 // Set to 1 if L/R and F/R signals from RC need to be
mixed
#define Leftcenter 1500 // when RC inputs are centered then input
should be 1.5mS
#define Rightcenter 1500 // when RC inputs are centered then input
should be 1.5mS
#define RCdeadband 35 // inputs do not have to be perfectly centered
to stop motors
#define scale 12 // scale factor for RC signal to PWM

//== BATTERY CHARGER SETTINGS ==
#define batvolt 487 // This is the nominal battery voltage reading.
Peak charge can only occur above this voltage.
#define lowvolt 410 // This is the voltage at which the speed control-
ler goes into recharge mode.
#define chargetimeout 300000 // If the battery voltage does not change
in this number of milliseconds then stop charging.

//== H BRIDGE SETTINGS ==
#define Leftmaxamps  330 // set overload current for left motor
#define Rightmaxamps 330 // set overload current for right motor
#define overloadtime 100 // time in mS before motor is re-enabled af-
ter overload occurs

//== SERVO SETTINGS ==
#define DServo0 1500 // default position for servo0 on "power up" -
1500uS is center position on most servos
#define DServo1 1500 // default position for servo1 on "power up" -
1500uS is center position on most servos
#define DServo2 1500 // default position for servo2 on "power up" -
1500uS is center position on most servos
#define DServo3 1500 // default position for servo3 on "power up" -
1500uS is center position on most servos
#define DServo4 1500 // default position for servo4 on "power up" -
1500uS is center position on most servos
#define DServo5 1500 // default position for servo5 on "power up" -
1500uS is center position on most servos
#define DServo6 1500 // default position for servo6 on "power up" -
1500uS is center position on most servos
```

APÊNDICE D - TUTORIAIS

TUTORIAL 1 - INSTALAÇÃO SISTEMA OPERACIONAL UBUNTU NA PANDA BOARD

Requisitos

Para prosseguir com o processo de instalação, é necessário que os seguintes itens estejam disponíveis:

- Panda Board, para completar a instalação do sistema operacional (SO),
- SD Card com ao menos 8 Gb, para instalação do SO com o ROS,
- Computador com Ubuntu e leitor de cartão SD,
- Cabo USB/Serial, para poder prosseguir a instalação do SO, e um
- Cabo USB/USB mini, para ligar a Panda Board à energia.

Passos

Adquirindo a Imagem do Sistema Operacional

Inicialmente deve-se adquirir a imagem que será gravada, neste caso, Ubuntu Precise Pangolin 12.04 LTS, sendo que esta pode ser adquirida através do site:

http://cdimage.ubuntu.com/releases/12.04/release/

Deve-se escolher a versão para o processador ARM OMAP 4. O caminho do arquivo que deve ser copiado é:

http://cdimage.ubuntu.com/releases/12.04/release/ubuntu-12.04-preinstalled-server-armhf+omap4.img.gz

Inicializando o cartão de memória

Após concluir o download da imagem, se o cartão SD for novo, deve-se criar uma nova tabela de partições no cartão. Para isso, pode ser utilizado o programa *gParted*, o qual pode ser instalado no sistema *Ubuntu* através do comando:

```
$ sudo apt-get install gparted
```

Com o programa instalado e inicializado, basta checar qual a unidade deve ser selecionada, provavelmente algo no formato /dev/sdX, onde X pode ser qualquer letra; caso só haja um disco rígido no computador, a unidade receberá a letra “b”, e portanto, é identificado por /dev/sdb.

Após selecionar a unidade correta, basta escolher a opção para criar a tabela de partições.

Se o cartão já foi inicializado alguma vez, provavelmente já estará previamente particionado. Pode-se utilizar o *gParted* para apagar as antigas partições. Para isto, basta selecionar a unidade correta e a partição que se deseja remover e, em seguida, clicar no ícone “apagar”.

Copiando a Imagem para o cartão de memória

Se a imagem do *Ubuntu* foi copiada para o diretório padrão (ie.: /home/<usuário>/Downloads), deve-se adentrar este diretório para prosseguir:

```
$ cd ~/Downloads
```

Para que a imagem seja gravada no cartão SD, deve-se descomprimir a imagem, o que pode ser feito através do comando:

```
$ gunzip ubuntu-12.04-preinstalled-desktop-armhf+omap4.img.gz
```

Após a extração da imagem, estará disponível um arquivo de mesmo nome, porém sem a extensão “gz” ao final. Com isso, basta escrever os dados no cartão SD, utilizando o comando; onde /dev/sdX é o nome da unidade de SD:

```
$ sudo dd bs=4M if=ubuntu-12.04-preinstalled-desktop-armhf+omap4.img
of=/dev/sdX
```

Após a gravação da imagem no cartão SD, deve-se garantir que o processo realmente já copiou todos os dados para ele através do comando:

```
$ sudo sync
```

Isso fará com que os dados da memória sejam sincronizados com os do cartão SD.

Instalação

Para prosseguir com o procedimento de instalação, será necessário inserir o cartão SD na *Panda Board* e a utilização de uma ferramenta adicional, que pode ser obtida através do comando:

```
$ sudo apt-get install minicom
```

O *minicom* é capaz de se comunicar com um terminal remoto através de uma interface serial, ou, como neste caso, uma interface usb/serial.

Neste momento, deve-se conectar o cabo conversor usb/serial no computador para proceder com a instalação. Inicialmente deve-se identificar qual foi a interface associada ao cabo; para isto basta executar o comando:

```
$ dmesg | grep -e tty
```

Com isto, serão exibidos as linhas de dados do log do Kernel Linux em que a palavra *tty* está presente. Deve ser observada uma mensagem similar à esta:

```
...USB Serial Device converter now attached to ttyXXX
```

Esta extensão pode variar de acordo com o fabricante do conversor, mas comumente é algo como ttyUSBX ou ttySX, onde X pode assumir qualquer valor numérico, geralmente 0. Identificando corretamente a interface, pode-se iniciar a aplicação *minicom*:

```
$ sudo minicom -s
           +-----[configuration]------+
           | Filenames and paths      |
           | File transfer protocols  |
           | Serial port setup        |
           | Modem and dialing        |
           | Screen and keyboard      |
           | Save setup as dfl        |
           | Save setup as..          |
           | Exit                     |
           | Exit from Minicom        |
           +--------------------------+
//Selecione "Serial port setup"
//Selecione A e use o tty associado anteriormente
//Exemplo: PandaBoard = /dev/ttyUSB0
//Selecione F para desligar o "Flow Control"
+-----------------------------------------------------------------------+
| A -     Serial Device      : /dev/ttyUSB2                              |
| B - Lockfile Location      : /var/lock                                 |
| C -   Callin Program       :                                           |
```

```
| D -  Callout Program      :                                       |
| E -    Bps/Par/Bits       : 115200 8N1                            |
| F - Hardware Flow Control : No                                    |
| G - Software Flow Control : No                                    |
|                                                                   |
|    Change which setting?                                          |
+-------------------------------------------------------------------+
//Pressione enter, salve como default e saia
//Com a conexão do modem inicializada, pressione Ctrl-A, depois Z e em
//seguida M
//Isto listará os menus disponíveis
+-------------------------------------------------------------------+
|                     Minicom Command Summary                       |
|                                                                   |
|          Commands can be called by CTRL-A <key>                   |
|                                                                   |
|               Main Functions                  Other Functions     |
|                                                                   |
| Dialing directory..D  run script (Go)....G | Clear Screen.......C  |
| Send files.........S  Receive files......R | cOnfigure Minicom..O  |
| comm Parameters....P  Add linefeed.......A | Suspend minicom....J  |
| Capture on/off.....L  Hangup.............H | eXit and reset.....X  |
| send break.........F  initialize Modem...M | Quit with no reset.Q  |
| Terminal settings..T  run Kermit.........K | Cursor key mode....I  |
| lineWrap on/off....W  local Echo on/off..E | Help screen........Z  |
| Paste file.........Y                       | scroll Back........B  |
|                                                                   |
|      Select function or press Enter for none.                     |
|                                                                   |
|             Written by Miquel van Smoorenburg 1991-1995           |
|             Some additions by Jukka Lahtinen 1997-2000            |
|             i18n by Arnaldo Carvalho de Melo 1998                 |
+-------------------------------------------------------------------+
//Uma vez concluída a configuração do terminal minicom, os dados da
serial serão exibidos
```

Após a configuração da serial estar concluída, pode-se conectar a Panda Board ao computador para troca de dados, através do cabo USB/Serial e alimentar a placa através do cabo USB/mini-USB, com isso, os dados de inicialização da Panda Board serão exibidos:

```
Texas Instruments X-Loader 1.41 (Jul 26 2010 - 19:26:00)
mmc read: Invalid size
Starting OS Bootloader from MMC/SD1 ...


U-Boot 1.1.4-L24.6-dirty (Jul 27 2010 - 12:02:16)

Load address: 0x80e80000
DRAM:  512 MB
Flash:  0 kB
```

```
In:     serial
Out:    serial
Err:    serial
Net:    KS8851SNL
Hit any key to stop autoboot:  0
mmc read: Invalid size
3351176 bytes read
## Booting image at 80500000 ...
   Image Name:   Ubuntu Kernel
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3351112 Bytes =  3.2 MB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
OK
Starting kernel ...
```

A imagem copiada para a *Panda Board* irá expandir, utilizando todo o espaço disponível no cartão. Com isso, todos os passos iniciais de configuração do SO estão concluídos.

Referências

https://wiki.ubuntu.com/ARM/OmapDesktopInstall

http://omappedia.org/wiki/Minicom

http://softswagen.wordpress.com/2013/02/21/how-to-install-ubuntu-12-04-lts-on-pandaboard-es/

https://wiki.ubuntu.com/ARM/QA/Pandaboard

TUTORIAL 2 – CONFIGURAÇÃO WI-FI PANDA BOARD

Requisitos

Para prosseguir com o processo de instalação, é necessário que os seguintes itens estejam disponíveis:

- Pandaboard,
- Cabo Usb/Serial,
- Cabo USB/mini-USB,
- Cabo Ethernet, e
- Cartão de Memória SD, com imagem Ubuntu 12.04 gravada.

Passos

Primeiramente deve-se ligar o cabo USB/Serial e o cabo RJ-45 à *Panda Board* e inicializar o minicom, o que pode ser feito através de um terminal linux com o comando:

```
$ sudo minicom -s
```

Insere-se então o cartão de memória na placa e liga-se o cabo USB/mini-USB para alimentação da placa. Inicialmente será carregado o *bootloader* da placa e descompactar-se-á o sistema na placa, agora ocupando todos os setores do disco, e em seguida o sistema será inicializado.

Na primeira inicialização do sistema serão necessários seguir alguns passos de configuração para que o sistema possa ser utilizado; isto inclui idioma, teclado, rede, usuário e grupos de pacotes (Sistema Base e Servidor SSH) que devem ser instalados.

Após estes passos serem concluídos, o sistema será carregado e o usuário poderá se logar. O primeiro passo é aproveitar a conexão com a rede cabeada e então atualizar os pacotes do sistema, o que pode ser feito através da sequência de comandos:

```
$ sudo apt-get udpate
$ sudo apt-get upgrade
```

Após concluir a atualização dos pacotes, o sistema deve ser reiniciado, para que assim seja possível carregar o Kernel atualizado. Quando o sistema estiver acessível novamente, pode-se iniciar a instalação e configuração da rede Wi-Fi; para tanto, faz-se necessário instalar algumas aplicações, caso ainda não estejam disponíveis:

```
$ sudo apt-get install wpa_supplicant wireless_tools
```

Após a instalação estar concluída, deve-se configurar alguns parâmetros de rede. Por garantia, alguns valores devem ser checados antes de se prosseguir com a instalação:

```
$ cat /etc/lsb-release
```

Com isso pode-se verficar a versão instalada do sistema operacional (SO), neste caso *DISTRIB_RELEASE = 12.04*. O kernel do SO também deve ser verificado; deve-se estar com, ao menos, a versão 3.2.0-1412-omap4 instalada. Estes dados podem ser obtidos através da execução do comando:

```
$ uname -arv
Linux heroes 3.2.0-1412-omap4 #16-Ubuntu SMP PREEMPT Tue Apr 17
19:38:42 UTC 2012 armv7l armv7l armv7l GNU/Linux
```

Tendo verificado estes dados, deve-se então iniciar o procedimento de configuração da rede Wi-Fi. Neste tutorial será tratada uma rede Wi-Fi configurada com *WPA2 Passphrase*.

Inicialmente verifica-se qual a interface associada com a Wi-Fi, cujos dados podem ser obtidos através da execução do comando:

```
$ iwconfig
lo        no wireless extensions.

wlan0     IEEE 802.11abgn  ESSID:"GVT-CC65"
          Mode:Managed  Frequency:2.462 GHz  Access Point:
6C:2E:85:E9:CC:69
          Bit Rate=18 Mb/s   Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
          Link Quality=65/70  Signal level=-45 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:3   Missed beacon:0

eth0      no wireless extensions.
```

A informação importante com a execução deste comando é a identificação da interface, neste caso wlan0. Em seguida pode-se verificar quais as redes wi-fi disponíveis, bastando para isto digitar o seguinte comando:

```
$ sudo iwlist wlan0 scan
wlan0     Scan completed :
          Cell 01 - Address: 7C:4F:B5:EC:67:1D
                    Channel:11
                    Frequency:2.462 GHz (Channel 11)
                    Quality=18/70  Signal level=-92 dBm
                    Encryption key:on
                    ESSID:"GVT-671E"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6
Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s
                    Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
                    Extra:tsf=000000413abe6497
                    Extra: Last beacon: 1656ms ago
                    IE: Unknown: 00084756542D36373145
                    IE: Unknown: 010882848B960C121824
                    IE: Unknown: 03010B
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : CCMP TKIP
                        Authentication Suites (1) : PSK
                    IE: WPA Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (2) : TKIP CCMP
                        Authentication Suites (1) : PSK
                    IE: Unknown: 2A0100
                    IE: Unknown: 32043048606C
                    IE: Unknown: DD0900037F01010000FF7F
                    IE: Unknown: DD0A00037F04010000000000
          Cell 02 - Address: 6C:2E:85:E9:CC:69
                    Channel:11
                    Frequency:2.462 GHz (Channel 11)
                    Quality=65/70  Signal level=-45 dBm
                    Encryption key:on
                    ESSID:"GVT-CC65"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6
Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s
                    Bit Rates:24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
                    Mode:Master
                    Extra:tsf=0000000145babff6
                    Extra: Last beacon: 1664ms ago
                    IE: Unknown: 00084756542D43433635
                    IE: Unknown: 010882848B960C121824
                    IE: Unknown: 03010B
                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : CCMP
                        Pairwise Ciphers (1) : CCMP
                        Authentication Suites (1) : PSK
                    IE: Unknown: 2A0100
                    IE: Unknown: 32043048606C
```

```
                    IE: Unknown:
DD180050F2020101840003A4000027A4000042435E0062322F00
                    IE: Unknown:
2D1A4C101BFF00000000000000000000000000000000000000000000
                    IE: Unknown:
3D160B0808000000000000000000000000000000000000000000
                    IE: Unknown: DD0900037F01010000FF7F
                    IE: Unknown: DD0A00037F04010000000000
                    IE: Unknown:
DD8B0050F204104A0001101044000102105700010010410001001003B00010310470010
6C2E85E9CC696C2E85E9CC696C2E85E91021000853616765 6D436F6D10230008466173
7432373634102400043132333410420004353637381054000800060050F20400011011
00174661737432373634 2D536167656D436F6D2D41502D36391008000200086103C0001
01
```

Assim pode-se identificar a rede desejada, neste caso GVT-CC65, e proceder com a configuração. Para que a interface consiga obter um endereço, deve-se ter em mãos o *password* codificado da rede, que pode ser obtido através da execução do comando:

```
$ wpa_passphrase <seu_ESSID> <seu_password>

Ex.:
$ wpa_passphrase GVT-CC65 teste_password
network={
        ssid="GVT-CC65"
        #psk="teste_password"

psk=46f69e37708e0daaf1350861c07e4155dddae4a97e2666f9cc5de3d27f139ab0
}
```

A informação que deve ser salva é o psk(negrito); com ela será possível autenticar e ingressar na rede. Edite o arquivo */etc/network/interfaces* e adicione os campos relacionados à rede Wi-Fi:

```
$ vim interface_wlan0

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp
```

```
# The wireless network interface
auto wlan0
iface wlan0 inet dhcp
wpa-driver wext
wpa-ssid <seu ESSID>
wpa-ap-scan 1
wpa-proto RSN
wpa-pairwise CCMP
wpa-group CCMP
wpa-key-mgmt WPA-PSK
wpa-psk <seu_PSK_codificado>
```

Basta sair do arquivo e então reiniciar as interfaces de rede:

```
$ sudo /etc/init.d/networking stop
$ sudo /etc/init.d/networking start
```

Para verificar se a interface conseguiu obter um IP na rede Wi-Fi, digite o comando:

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 0e:60:3b:0f:3c:0a
          inet6 addr: fe80::c60:3bff:fe0f:3c0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1488  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:250 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:85916 (85.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 0e:20:3b:0f:3c:0a
          inet addr:192.168.25.19  Bcast:192.168.25.255
Mask:255.255.255.0
          inet6 addr: fe80::c20:3bff:fe0f:3c0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1297 errors:0 dropped:0 overruns:0 frame:0
          TX packets:47 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:135977 (135.9 KB)  TX bytes:5144 (5.1 KB)
```

Se tudo estiver correto, a interface Wi-Fi(wlan0) receberá um IP, neste caso 192.168.25.19.

Referências

http://www.youtube.com/watch?v=MjcthIVHWTM&list=PLF505C0DA894CA58A

TUTORIAL 3 – INSTALAÇÃO ROS

Requisitos

Para prosseguir com o processo de instalação, é necessário que os seguintes itens estejam disponíveis:

- Panda Board,
- Cabo Usb/Serial,
- Cabo USB/mini-USB,
- Cabo Ethernet, e
- Cartão de Memória SD, com imagem Ubuntu 12.04 gravada, instalada e configurada.

Passos

Antes de proceder com a instalação do ROS é necessário verificar quais são os repositórios do *Ubuntu* que estão disponíveis para a plataforma. O recomendável é que os repositórios *restricted*, *universe* e *mutiverse* estejam disponíveis. Para isto basta verificar o arquivo localizado em */etc/apt/source.list* através do comando:

```
$ sudo nano /etc/apt/source.list
```

As linhas contendo os repositórios não devem estar comentadas, ou seja, sem o # no início da linha.
Caso seja necessário descomentar as linhas, deve-se salvar o arquivo e em seguida executar o comando:

```
$ sudo apt-get update
```

Assim a lista de aplicativos do repositório será atualizada.

Durante o desenvolvimento deste trabalho, o site do ROS disponibilizava pacotes binários para a versão 12.04 Precise Pangolin no formato armhf (ARM – Hard Float). Para adicionar este repositório ao *Ubuntu*, basta digitar o seguinte comando:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ahendrix-mirror/ubuntu
precise main" > /etc/apt/sources.list.d/ros-latest.list'
```

Isto fará com que o caminho para um novo repositório seja adicionado ao arquivo */etc/apt/sources.list.d/ros-latest.list*; caso o arquivo já exista, o mesmo será sobrescrito.

Antes de atualizar os repositórios e instalar o ROS, deve-se adicionar a chave do repositório desejado às chaves confiáveis do SO, utilizando-se o comando:

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Instalação Pacotes Básicos

Após adicionar um novo repositório ao sistema deve-se atualizar a lista de aplicativos, antes de proceder com a instalação, para que os pacotes do ROS sejam listados como aplicações válidas:

```
$ sudo apt-get update
```

Após concluída a atualização dos repositórios, deve-se instalar o pacote do ROS desejado, neste caso:

```
$ sudo apt-get install ros-groovy-ros-base ros-groovy-rosserial ros-
groovy-rosserial-arduino
```

O ROS possui um sistema que permite a resolução de dependências de sistema antes da compilação de componentes do ROS. Para isso deve-se executar o comando:

```
$ sudo rosdep init
$ rosdep update
```

Setup do Ambiente

Por fim, para que os aplicativos do ROS sejam facilmente acessados através da linha de comando, basta executar o seguinte comando:

```
$ echo "source /opt/ros/groovy/setup.bash" >> ~/bashrc
$ source ~/.bashrc
```

O primeiro comando fará com que ao iniciar uma sessão do usuário os aplicativos e variáveis de ambiente do ROS sejam inicializadas. O segundo comando fará com que as variáveis de ambiente utilizadas pelo ROS sejam carregadas instantaneamente no terminal aberto.

Geração da biblioteca para o Arduino IDE

Para que a aplicação desenvolvida para o *Arduino* consiga se comunicar com o nodo que executará a interface com o ROS, deve-se gerar bibliotecas compatíveis, neste caso com a *Panda Board*. Para isto basta invocar a seguinte linha de comando:

```
$ rosrun rosserial_arduino make_libraries.py ~
```

Isso fará com que os arquivos da biblioteca sejam gerados na home do usuário, e.g. */home/<nome_do_usuario>/ros_lib*. Para continuar com o desenvolvimento será necessário copiar os arquivos para o computador em que a aplicação para o *Arduino* está sendo desenvolvida. Para isto recomenda-se a cópia do arquivo via cliente SFTP para o host.

A pasta contendo os arquivos de biblioteca do *rosserial* (ros_lib), deve ser copiada para a pasta "<sketchbook>/libraries" do *Arduino*, no seu SO, neste caso:

/home/<nome_do_usuário>/arduino/libraries
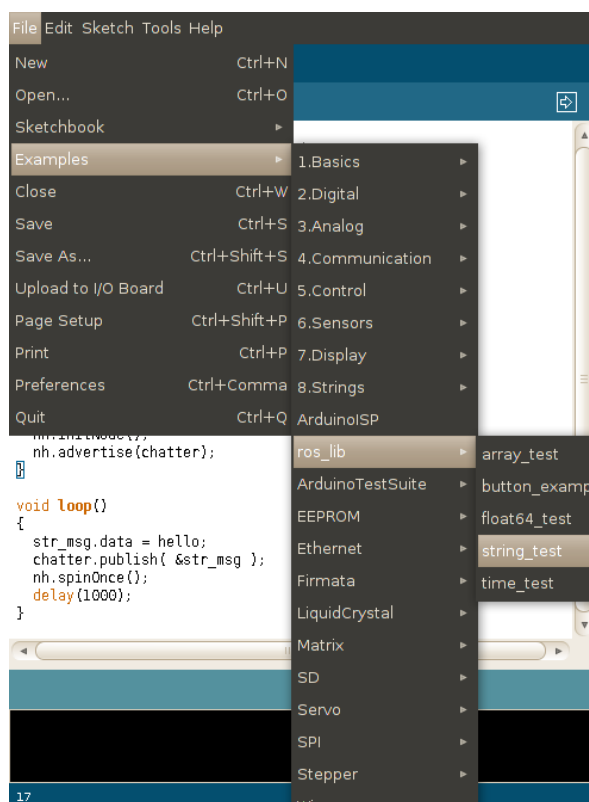
Figura 1 - Biblioteca corretamente instalada no Arduino IDE (Fonte:

http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup?action=AttachFile&do=get&target=arduino_ide_examples_screenshot.png)

Pode-se verificar se as biblioteca do ROS foi corretamente instalada, se os exemplos estiverem disponíveis, conforme figura 1. Após estar corretamente instalada, pode-se carregar os programas que gerenciam os movimentos do *Wild Thumper* nas placas Master no *Arduino Uno* e Slave no *Wild Thumper Controller.*

Referências

http://www.ros.org/wiki/groovy/Installation/Ubuntu

http://www.ros.org/wiki/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup

TUTORIAL 4 – MONTAGEM ELÉTRICA DO PROJETO

Requisitos

Para a montagem do Wild Thumper, deve-se ter disponível os seguintes materiais:

- Panda Board ligada a energia através de um cabo com ponta P4,
- Cabo USB-A/USB-B para ligação entre Panda Board e Arduino Uno, e
- Cabos Diversos para ligação de saídas e entradas das placas.

Passos

Figura 1 - Cabo USB A/USB B (Fonte: Autoria Própria).

Ligar a *Panda Board* ao *Arduino Uno* através do cabo USB A / USB B, o cabo pode ser visualizado na figura 1;

Figura 2 - Cabo para ligação I2C (Fonte: Autoria Própria).

Ligar os pinos A4 e A5 do *Arduino Uno* aos pinos A4 e A5 do *Wild Thumper Controller*, respectivamente, utilizando cabo conforme figura 2;

Figura 3 - Cabo utilizado para alimentar a Panda Board (Fonte: Autoria Própria).

Ligar o GND e o VCC da placa do *Wild Thumper Controller* ao pino P4 de alimentação da *Panda Board*, cabo similar ao da figura 3;

Ligar as saídas da ponte H aos motores do *Wild Thumper*, L_Motor e R_Motor. A sequência a partir da saída mais externa à placa do L_Motor deve seguir a coloração Preto, Vermelho, Vermelho e Preto.

A montagem elétrica do *Wild Thumper* deve seguir o esquemático da figura 1.
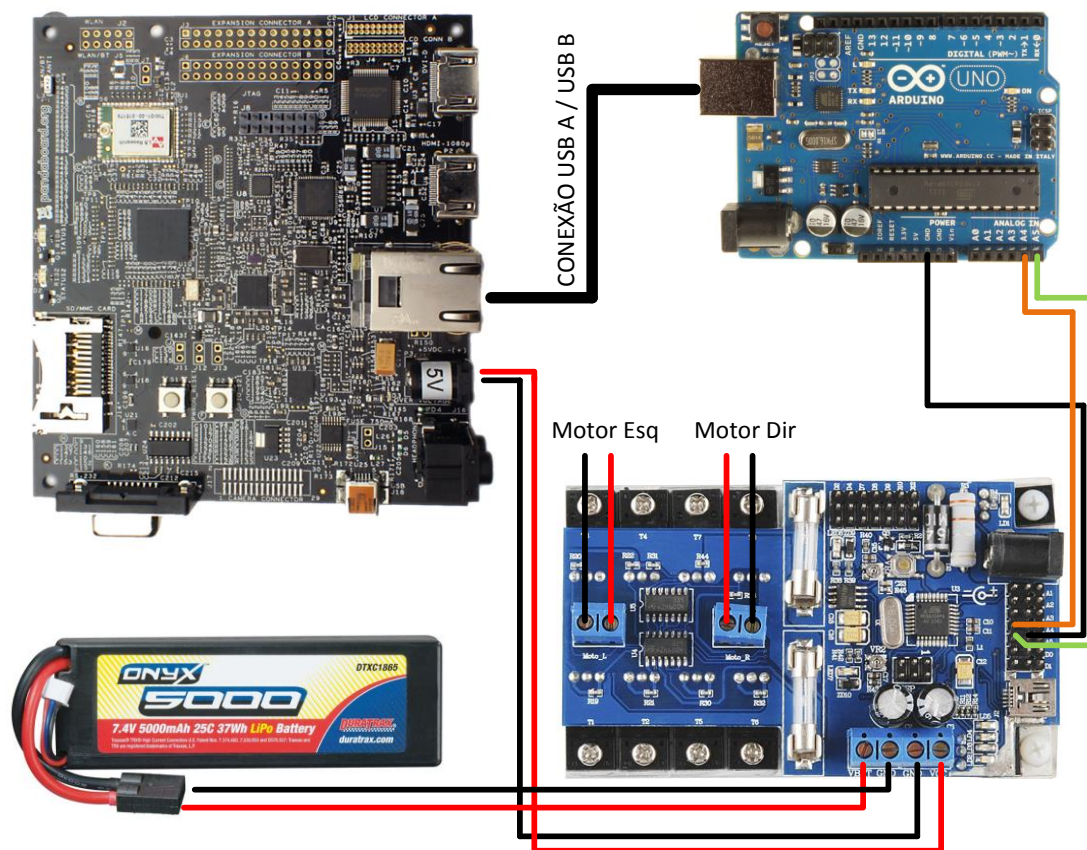
Figura 1 - Ligações elétricas para o funcionamento do Wild Thumper (Fonte: Imagem baseada em fontes diversas)

Referências

http://www.hobbytronics.co.za/content/images/thumbs/0002204_wild-thumper-controller-board.jpeg

http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front_450px.jpg

http://www.liquidware.com/system/0000/3733/PandaBoard_Top_2.jpg

http://a248.e.akamai.net/origin-cdn.volusion.com/vmhfv.fqdqc/v/vspfiles/photos/DTXC1865-2.jpg?1377874965

TUTORIAL 5 – COMANDOS BÁSICOS ROS

Requisitos

Para prosseguir com o tutorial será necessário:

- PC, Máquina Virtual ou Panda Board, executando o sistema operacional (SO) *Ubuntu,* com interface gráfica, e
- Framework Robot Operating System (ROS) instalado.

Conteúdo Apresentado

Neste tutorial serão apresentados comandos básicos do ROS, assim como exemplos de sua utilização. A vantagem no uso do ROS é a limitação a uma interface comum de comunicação e a formatos específicos de mensagens; com isso, diversos drivers, câmeras e aplicações podem se comunicar de forma concisa. A troca de mensagens no ROS é feita através do protocolo TCP/IP. Esta troca pode ser realizada seguindo o modelo *publisher/subscriber*, ou seja, a mensagem é publicada por um nodo, e consumida por um ou mais nós que assinam este tópico.

Para que dois ou mais nós, que a princípio não se conhecem, consigam trocar informações, é necessário que um agente comum aos dois realize esta interface. O ROS, para solucionar este problema, cria uma entidade que conhece todos os nós do sistema e, quando necessário, realiza a interface entre os nós. Para esta entidade é dado o nome de *core* do sistema. Por isso, para inicializar um nodo no ROS, é necessário que o *core* esteja rodando. Para lançar a aplicação, basta digitar em um terminal que possua acesso aos binários do ROS:

```
$ roscore
... logging to /home/taberu/.ros/log/3c1f1414-103c-11e3-b95c-
000c294ebc5d/roslaunch-ubuntu-1233.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:53569/
ros_comm version 1.9.47


SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion
```

```
NODES

auto-starting new master
process[master]: started with pid [1247]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 3c1f1414-103c-11e3-b95c-000c294ebc5d
process[rosout-1]: started with pid [1260]
started core service [/rosout]
```

Com a inicialização do core, algumas informações importantes são exibidas, como o endereço em que o core estará aguardando as requisições (ROS_MASTER_URI= http://ubuntu:11311). Se todos os nós forem executados localmente o endereço não precisa ser válido para a rede externa, neste caso http://ubuntu:11311 é um URI válido. Outra informação importante é a criação de alguns parâmetros contendo informações sobre a versão que está sendo executada, *rosdistro* e *rosversion*. Para listar os parâmetros que atualmente estão sendo executados pelo ROS, pode-se entrar o seguinte comando:

```
$ rosparam list
/rosdistro
/roslaunch/uris/host_ubuntu__53569
/rosversion
/run_id
```

Para acessar os dados contidos em um parâmetro basta digitar o comando:

```
$ rosparam get <parametro>

Ex.:
$ rosparam get /rosdistro
'groovy

  '
```

Para listar todas as opções de comandos disponíveis para o *rosparam*, basta digitar em um terminal:

```
$ rosparam help
rosparam is a command-line tool for getting, setting, and deleting
parameters from the ROS Parameter Server.

Commands:
        rosparam set    set parameter
```

```
        rosparam get    get parameter
        rosparam load   load parameters from file
        rosparam dump   dump parameters to file
        rosparam delete delete parameter
        rosparam list   list parameter names
```

Pode-se visualizar os nós conectados ao *roscore* através do comando:

```
$ rosnode list
/rosout
```

O acesso às informações de um nodo pode ser feito através do comando:

```
$ rosnode info <nodo>

Ex.:
$ rosnode info /rosout
----------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/set_logger_level
 * /rosout/get_loggers


contacting node http://ubuntu:55191/ ...
Pid: 1260
```

Assim informações de tópicos e serviços que o nodo assina ou publica serão exibidas no terminal. Para verificar quais são todos os comandos disponíveis pelo *rosnode*, pode-se digitar em um terminal:

```
$ rosnode help
rosnode is a command-line tool for printing information about ROS
Nodes.

Commands:
        rosnode ping    test connectivity to node
        rosnode list    list active nodes
        rosnode info    print information about node
```

```
        rosnode machine list nodes running on a particular machine or
list machines
        rosnode kill    kill a running node
        rosnode cleanup purge registration information of unreachable
nodes

Type rosnode <command> -h for more detailed usage, e.g. 'rosnode ping
-h'
```

Para a finalidade de teste, instale os tutoriais do ROS. Em um terminal digite:

```
$ sudo apt-get install ros-groovy-ros-tutorials
```

Após a instalação do pacote, inicialize três terminais.

No primeiro inicialize o core do ROS:

```
$ roscore
```

No segundo inicialize o pacote *turtlesim*, isto pode ser feito através do comando:

```
$ rosrun turtlesim turtlesim_node
```

Agora no terceiro terminal, liste todos os tópicos em execução:

```
$ rostopic list
...
/turtle1/command_velocity
...
```

Qualquer tópico ou parâmetro no ROS pode ser renomeado, o que pode ser feito através de uma opção na linha de comando da aplicação. Por exemplo, se o tópico /turtle1/command_velocity fosse renomeado para /turtle1/cmd_vel, teríamos que inicializar o *turtlesim_node* com:

```
$ rosrun turtlesim turtlesim_node
turtle1/command_velocity:=turtle1/cmd_vel
```

Ao listar novamente os tópicos, pode-se ver que o parâmetro foi renomeado:

```
$ rostopic list
...
/turtle1/cmd_vel
...
```

Referências

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

TUTORIAL 6 – EXECUÇÃO DO TRABALHO PRÁTICO

Requisitos

Para que seja possível executar o ambiente deve-se cumprir alguns passos:

1 – Ter concluído todos os procedimentos descritos ao menos até o tutorial 4 (Montagem do Wild Thumper);

2 – Inicializar a Panda Board;

3 – Verificar se conexão Wi-Fi está correta:

```
$ ifconfig
wlan0     Link encap:Ethernet  HWaddr 0e:20:3b:0f:3c:0a
          inet addr:192.168.25.29  Bcast:192.168.25.255
Mask:255.255.255.0
          inet6 addr: fe80::c20:3bff:fe0f:3c0a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2735 (2.7 KB)  TX bytes:3411 (3.4 KB)
```

Passos PandaBoard

Se a wi-fi estiver ativa e funcionando, pode-se iniciar a aplicação de controle do Wild Thumper.

1 – Deve-se exportar a variável ROS_IP com o IP recebido pelo DHCP na rede, no caso 192.168.25.29, o que fará com que a aplicação do ROS que controla os movimentos do carrinho conecte-se ao *core* através da rede.

```
$ export ROS_IP=<ip_dhcp>

Ex.:
$ export ROS_IP=192.168.25.29
```

2 – Inicializar o roscore para gerência dos nós:

```
$ roscore
```

3 – Atribuir valor ao parâmetro *robot/name* para que o core seja identificado pela aplicação android de controle do carrinho:

```
$ rosparam set robot/name WildThumper
```

4 – Atribuir valor ao parâmetro *robot/type* para que o core seja identificado pela aplicação android:

```
$ rosparam set robot/type turtlebot
```

5 – Inicializar nodo de comunicação serial com Arduino com remapeamento de tópico, ou seja, renomeando o tópico *WildThumper/teleop* para *turtlebot_node/cmd_vel*. Assim será possível receber as informações geradas pela aplicação android:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
WildThumper/teleop:=turtlebot_node/cmd_vel
```

Passos Celular

Este é o último passo para execução do controle sobre o Wild Thumper. Inicialmente deve-se instalar a aplicação no sistema android que será utilizado (ie. Smartphone, Tablet), que pode ser visualizada na figura 1. A aplicação é o ROS Teleop, da Willow Garage:

https://play.google.com/store/apps/details?id=ros.android.teleop&hl=pt_BR

Quando inicializar a aplicação, será necessário configurar o endereço em que o *core* do ROS está sendo executado, no formato:

http://<ip_dhcp>:11311

Ex.:
Neste caso o endereço exportado pela variável de ambiente ROS_IP ser utilizada:

http://192.168.25.29:11311

Com isso a aplicação será inicializada e estará conectada com o mesmo core em que o Wild Thumper está ligado. Será apresentada uma lista com os nós disponíveis e que são compatíveis com o Teleop. Ao selecionar o robô, será apresentada uma tela com

dois campos, mais à esquerda da tela é apresentada a imagem da câmera, se disponível, e a esquerda da tela um plano cartesiano, utilizado para geração dos tópicos de controle de movimentação.
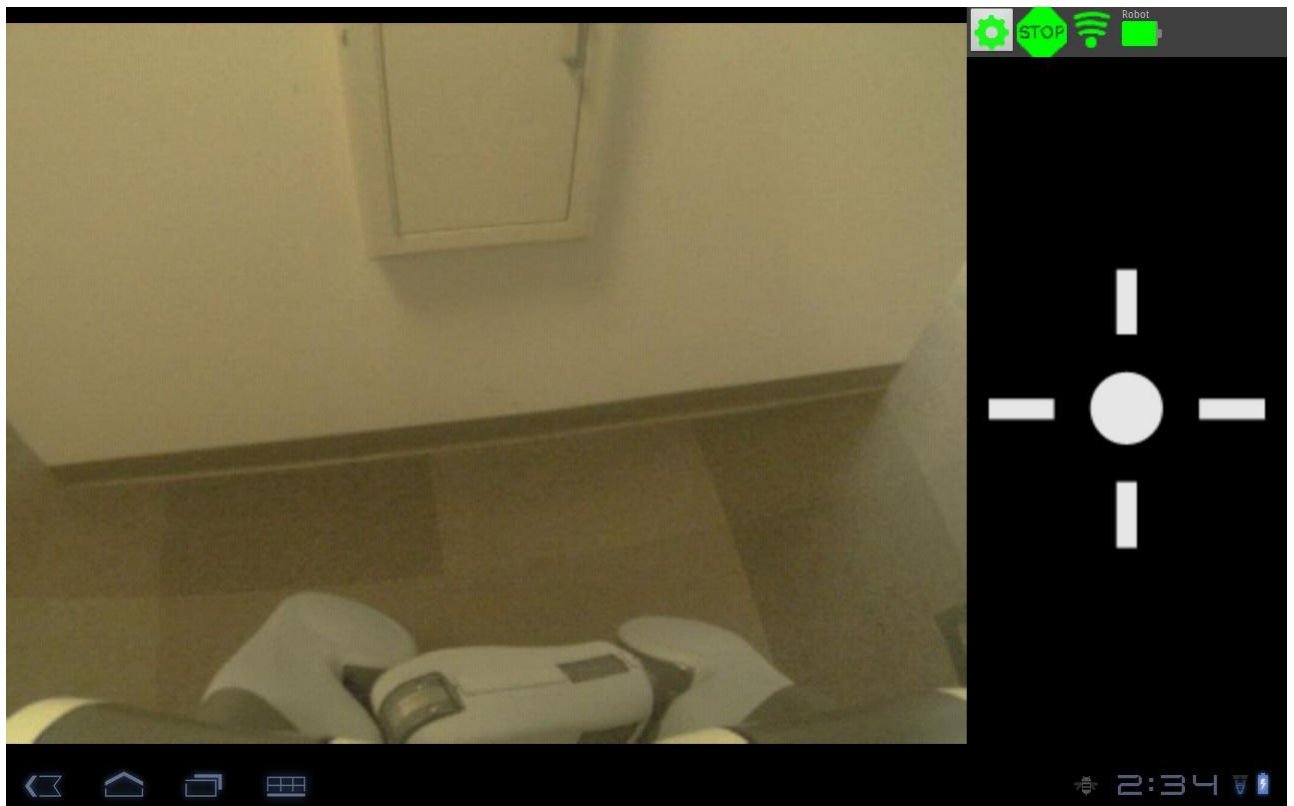
Figura 16 - Imagem aplicação Android para tele operação (Fonte: https://lh4.ggpht.com/63VgIrSC4S-do6x4bYVIZAjNet1gWvImGYXI6LdGKhF61NMLMaSHEDPpFO9xHgk8tA=h900).